INTERWOVEN

DataDeploy™ Administration Guide

Release 5.5.1

INTERWOVEN

Interwoven, Inc.
803 11th Ave.
Sunnyvale, CA 94089
http://www.interwoven.com
Printed in the United States of America
Release 5.5.1
Part # 40-00-20-25-00-551-300

Table of Contents

About This Book

The *DataDeploy Administration Guide* describes how to install, configure, and use DataDeploy™ with TeamSite® and TeamSite Templating.

This guide is primarily intended for Web server administrators and system administrators. Users of this manual should be familiar with basic UNIX® or Windows® commands (as appropriate) and be able to use a text editor such as emacs or vi (on UNIX) or Notepad (on Windows NT® or Windows 2000). Many of the operations described in this manual require UNIX root or Windows Administrator access to the TeamSite server.

Typographical and Notation Conventions

This section describes the following typographical and notation conventions:

- Typographical Conventions
- iw-home and dd-home Notation on Solaris™ and Windows Systems
- Notation Conventions for Directory Paths

Typographical Conventions

This manual uses the following notation conventions:

Convention	Definition and Usage
Bold	Text that appears in a GUI element (for example, a menu item, button, or element of a dialog box) and command names are shown in bold. For example: Click **Edit File** in the Button Bar.

Convention	Definition and Usage
Italic	Book titles appear in italics. Terms are italicized the first time they are introduced. Important information may be italicized for emphasis.
`Monospace`	Commands, command-line output, and file names are in monospace type. For example: The `iwextattr` command-line tool allows you to set and look up extended attributes on a file.
`Monospaced italic`	Monospaced italics are used for command-line variables. For example: `iwckrole` *`role user`* This means that you must replace *`role`* and *`user`* with your values.
`Monospaced bold`	Monospaced bold represents information you enter in response to system prompts. The character that appears before a line of user input represents the command prompt, and should not be typed. For example: **`iwextattr -s project=proj1 //IWSERVER/default/main/dev/WORKAREA/andre/products/index.html`**
`Monospaced bold italic`	Monospaced bold italic text is used to indicate a variable in user input. For example: `%` **`iwextattr -s project=`*`projectname workareavpath`* means that you must insert the values of *`projectname`* and *`workareavpath`* when you enter this command.
`[]`	Square brackets surrounding a command-line argument mean that the argument is optional.
`|`	Vertical bars separating command-line arguments mean that only one of the arguments can be used.

iw-home and dd-home Notation on Solaris and Windows Systems

iw-home

This manual does not use the Solaris notation (`iw-home`; note the lack of italics) except when specifically referring to procedures performed only in Solaris (as in the Solaris section of "Installation Procedures" on page 19).

This manual uses the Windows version of `iw-home` notation (*`iw-home`*) when discussing both Solaris™ and Windows NT or Windows 2000 systems. The italics are an Interwoven convention identifying *`iw-home`* as a variable. You should interpret the *`iw-home`* notation used in this manual as follows:

- On Solaris systems, *`iw-home`* is the literal name of the directory containing the TeamSite program files.
- On Windows systems, *`iw-home`* is the symbolic name of the directory that contains your TeamSite program files. The default value of *`iw-home`* on Windows systems is:

  ```
  C:\Program Files\Interwoven\TeamSite
  ```

  The administrator performing Windows installation may have chosen an installation directory different from the default.

dd-home

Certain files that you will need to edit are located in a directory called *`dd-home`*. The italics are an Interwoven convention identifying *`dd-home`* as a variable. You should interpret the *`dd-home`* notation used in this manual as follows:

On both Windows and Solaris systems, *`dd-home`* is the symbolic name of the directory containing various DataDeploy files. The default value of *`dd-home`* is different, depending on your installation scenario. See "Installation Procedures" on page 19 to review various default values.

Notation Conventions for Directory Paths

This manual uses UNIX conventions for directory paths because this is the convention followed in all Interwoven cross-platform documentation. These conventions mandate using forward slashes (/). For example:

```
docroot/news/front.html
```

On a Windows system, you would type such a path with backward slashes:

```
docroot\news\front.html
```

This manual only uses Windows conventions for paths when referring to a Windows-specific directory.

Editing Text on Windows Systems

It is recommended that you use WordPad rather than Notepad to edit text on a Windows system. Other text editors may also be used.

Online Documentation Errata

Additions and corrections to this document are available in PDF format at the following site. Browse to the download and release notes directories.

```
http://support.interwoven.com
```

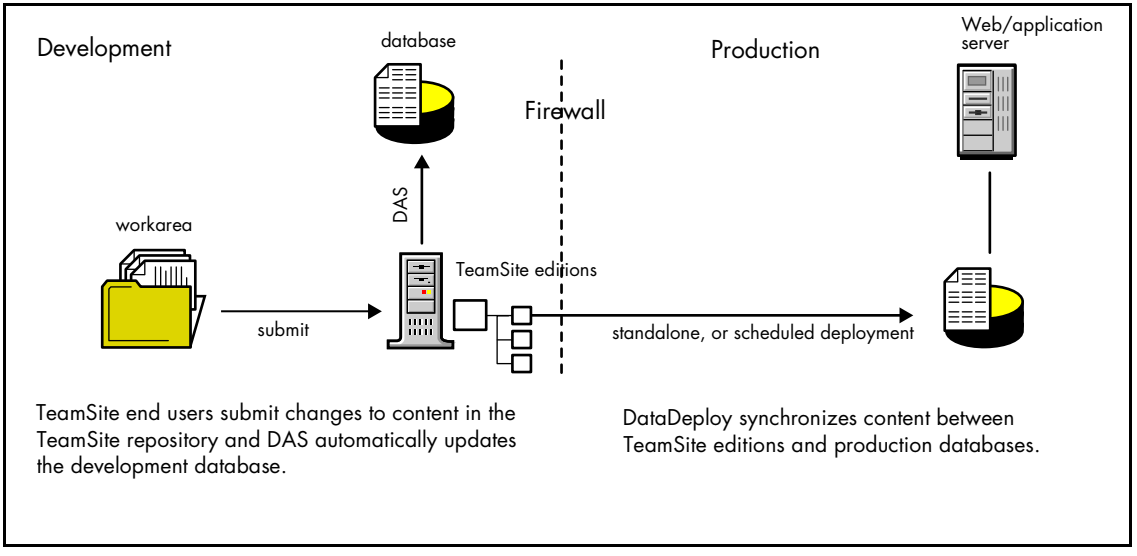
Chapter 1

Introduction

Welcome, and congratulations on using Interwoven DataDeploy software! DataDeploy is industry-leading distribution software that enables enterprises to distribute content from Interwoven® TeamSite repositories to industry-standard relational databases. DataDeploy is a powerful and flexible tool for supporting your Web development and production environments.

In development environments, the DataDeploy Database Auto-Synchronization (DAS) feature enables you to automatically update databases when changes are made to content in the TeamSite repository. XML content and TeamSite metadata can be deployed to tables in relational databases that are customized based on the content being deployed.

Additionally, you can use DataDeploy to synchronize data between Teamsite editions and development databases across a firewall. You can perform deployments manually from the DataDeploy administration graphical user interface (GUI) or command line, or you can schedule deployments if DataDeploy is used with OpenDeploy.



Case Study: Acme Corp.

In this section we introduce a fictional company named *Acme*, which has recently adopted a suite of Interwoven products for their enterprise-wide content infrastructure platform. How Acme chooses to use DataDeploy in conjunction with the other products in the suite illustrates how DataDeploy can become a key part of your organization's content infrastructure.

Acme is headquartered in San Francisco with regional centers in Paris, Munich, and Tokyo. Development and production environments at Acme are heterogeneous: they use a mix of Windows and UNIX Web servers, and databases from different vendors.

Acme maintains customer Web sites that are localized for each region. Acme's customer Web sites feature a large product catalog which is frequently updated, sometimes multiple times a day. Currently, much of the content and application code for the catalog is developed at the home office in San Francisco. Regional offices are responsible for making changes suitable for the Web site in their area to the content they receive from the San Francisco office.

Acme plans to use DataDeploy to meet the following needs:

- Acme uses a variety of industry-standard databases. Furthermore, databases in Paris, Munich, and Tokyo run in the language locale of their respective region. Acme needs a distribution solution that can deploy multibyte content across different platforms into localized databases.
- Web application developers need to test their applications with up-to-date content from the database. To provide that content, databases must be updated when changes to content in the TeamSite repository are submitted.
- To automate the process of synchronizing TeamSite content with database content, Acme needs the following:
  - Content in a structured form (XML files), or metadata about that content.
  - A way to create database schemas that are based on that structured content, or *user-defined schemas*.
  - A reliable, flexible, and scalable way of triggering deployments.
- Acme wants to dynamically enhance some product information during deployments. For example, Acme wants price information to be transformed from U.S. dollars to the equivalent price in local currency when such data is deployed to databases in the regional offices.

- Because DataDeploy will be used at several sites and by administrators with varying levels of experience, Acme needs a product that is easy to install and configure.
- Acme will be deploying data outside the protection of firewalls. They need a solution that ensures that deployed content will remain secure.

The DataDeploy Advantage

DataDeploy offers the following features that met Acme's needs:

- DataDeploy is an internationalized product that supports the deployment of multibyte content to non-U.S. English databases.

  See Appendix D, "Internationalization" for details about that feature.

- The DataDeploy DAS module automatically synchronizes content between TeamSite areas (workareas and staging areas) and development databases. When Acme content contributors make changes to content in their TeamSite areas those changes are immediately reflected in the development database. Acme's Web application developers can test their applications with that up-to-date information.

  See Chapter 8, "Automating Deployment with DAS" for details about that feature.

- Acme content contributors use TeamSite Templating to add content to the TeamSite repository. They also frequently use the TeamSite set metadata feature to "tag" content for later search and reuse.

  In TeamSite Templating, end users create *data content records*, special XML files that store content. Those files are created by filling in forms that are based on *data capture templates*, which are also XML files. Data capture templates also define the forms that are used to create metadata.

  Acme uses many different data capture templates. Using the DataDeploy administration GUI, Acme can easily map those templates to user-defined schemas, then use DAS to streamline the synchronization of data between the TeamSite repository and the development database.

  See "Deploying Data to User-Defined Database Schemas: Overview" on page 49 for details about that feature.

- Acme can then use DAS to streamline the synchronization of data between the TeamSite repository and the development database. Furthermore, Acme can take advantage of the TeamSite Event Server component to selectively specify the TeamSite events that will trigger DAS deployments.

  See Appendix C, “Event Server” for details about that feature.
- The DataDeploy tuple preprocessing feature enables Acme to dynamically enhance data before it is being deployed. For example: Acme headquarters updates the prices for its products and creates a new edition of their Web site. The data from that edition is deployed to production databases at the regional offices. During the deployment process, DataDeploy dynamically translates the U.S. prices into the currency of the target databases’ region.

  See “Enhancing Data Before Deployment” on page 93 for details about that feature.
- DataDeploy can be administered through an easy-to-use Web-based graphical user interface.

  See the *DataDeploy User’s Guide* for details about that feature.
- DataDeploy can be integrated with OpenDeploy to provide secure, transactional deployments of encrypted content to production database by using Secure Socket Layer (SSL). Additionally, DataDeploy leverages OpenDeploy’s content rollback capabilities. If a deployment fails, database content is rolled back to a previous version.

  See Chapter 7, “Synchronizing OpenDeploy and DataDeploy” for details about that feature.

The features described above illustrate some of the most common ways in which DataDeploy is used. DataDeploy also includes the following capabilities which you might find useful:

- Deploy source data to XML files
- Filter data during deployments
- Execute SQL queries
- Deploy incremental differences between two TeamSite editions (when integrated with OpenDeploy)

The next chapter describes the installation of DataDeploy, and the two subsequent chapters describe basic DataDeploy concepts.

Chapter 2

Installation

This chapter describes DataDeploy setup options and leads you through the tasks required to install the software.

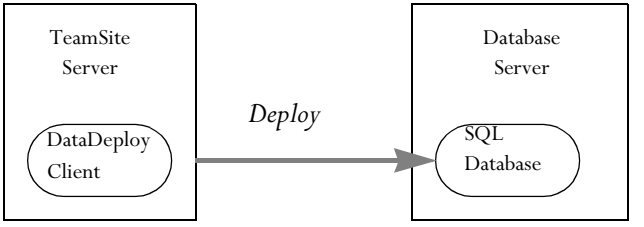
Refer to the *DataDeploy Release Notes* for the latest information on the following installation-related topics:

- Supported operating systems
- Localized operating system support
- RAM requirements
- Storage requirements
- Patch requirements
- Supported browsers
- Compatibility between DataDeploy releases
- Compatibility between DataDeploy and other Interwoven products

Client/Server Setup Options

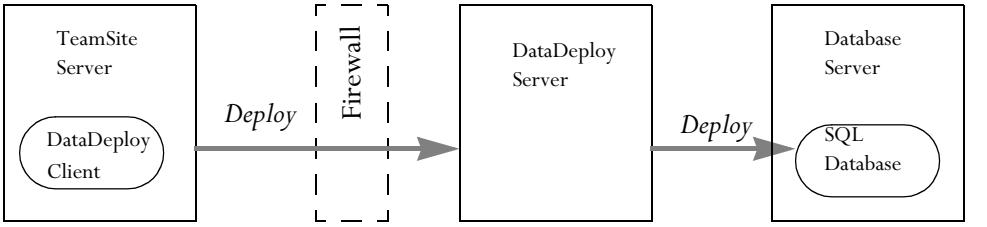
When deploying to a database, you can set up DataDeploy to operate in either a *two-tier* or *three-tier* architecture.

Two-tier architecture incorporates two systems: the TeamSite server host machine that executes the DataDeploy client and an application server containing the SQL database. The application server can be any server on the network (such as the production Web server, although this is not a system requirement). Two-tier systems are typically used at sites that do not require firewall protection between the TeamSite server and the application or production server.



Two-Tier Architecture

Three-tier architecture incorporates a third system acting as a DataDeploy server. Three-tier systems are typically used at sites requiring firewall protection between the TeamSite server and the application or production Web server. In this scenario, the TeamSite server does not directly connect to the database server; instead, it connects to the DataDeploy server, which then connects to the database server. You cannot use DataDeploy's DAS feature when using a three-tier architecture.



Three-Tier Architecture

Running the DataDeploy Daemon as a Service

You can optionally configure the `Interwoven DataDeploy` service to start the DataDeploy daemon for three-tier operation or Database Auto-Synchronization (DAS) operation. When set up for three-tier operation, the DataDeploy daemon takes its input from the `iwdd.ipl` command issued from the command line. When set up for DAS operation, the DataDeploy daemon takes its input from the `iwsyncdb.ipl` script that runs as part of DAS startup. See "Editing iwsyncdb.cfg" on page 175 for information about specifying DAS or three-tier operation. See "Invoking DataDeploy" on page 159 for details about executing `iwdd.ipl` from the command line. See "Running iwsyncdb.ipl" on page 176 for details about `iwsyncdb.ipl`.

The `Interwoven DataDeploy` service automatically starts the DataDeploy daemon for DAS operation if the `iwsyncdb.cfg` file exists in *`dd-home`*`/conf`. If `iwsyncdb.cfg` does not exist, the `Interwoven DataDeploy` service starts the DataDeploy daemon for three-tier operation.

Installation Procedures

The following subsections describe installation on Solaris and Windows operating systems.

Solaris Systems

To install DataDeploy on a Solaris system:

1. Change user to `root`.
2. Unzip and untar the DataDeploy tar file:

   `gunzip < datadeploy.tar.gz | tar -xvpf -`

   where *`datadeploy.tar.gz`* is a variable representing the name of the compressed tar file containing the DataDeploy software.

   After you execute this command, a `datadeploy` directory and its associated subdirectories are created if they do not already exist.
3. Go to the `datadeploy` directory and execute the `startinstalldd` script.
4. Specify where you want to install DataDeploy.

5. If TeamSite and TeamSite Templating is installed on the system where you are installing DataDeploy, you are prompted to select whether you want to activate the DataDeploy administration graphical user interface.

After DataDeploy is installed, you may need to resynchronize the tracker table. Refer "Resynchronizing the Tracker Table" on page 21 to determine whether this procedure is necessary.

Windows Systems

To install DataDeploy on a Windows system (Windows NT or Windows 2000):

1. Download the DataDeploy bundle from its distribution media. If the file is zipped, unzip it.
2. Double click the DataDeploy bundle icon.
3. If TeamSite and TeamSite Templating is installed on the system where you are installing DataDeploy, you are prompted to select whether you want to activate the DataDeploy administration graphical user interface.
4. Select the location where you want to install DataDeploy.
5. Refer to the section "Resynchronizing the Tracker Table" on page 21 to determine whether you need to perform resychronize the tracker table.

After DataDeploy is installed, you may need to resynchronize the tracker table. Refer "Resynchronizing the Tracker Table" on page 21 to determine whether this procedure is necessary.

Setting Up DAS and Metadata Capture

If you will use metadata capture with DAS, do the following:

1. Rename the following file:
   dd-home`/conf/mdc_ddcfg.template.example`

   to:
   dd-home`/conf/mdc_ddcfg.template`.
2. Configure the appropriate database sections.
3. Ensure that you also set up *iw-home*`/local/config/datacapture.cfg`.

Resynchronizing the Tracker Table

After DataDeploy is installed, you must resynchronize the tracker table if both of the following are true:

- You are migrating from TeamSite 4.2.1 to 4.5.1.
- The directory structure from which you were deploying data contained subbranches.

Under these conditions, tracker table resynchronization is necessary to ensure that all tables remain synchronized if you delete a branch that contains subbranches. For example, resynchronization ensures that when you delete `branch1` from the following directory structure, DataDeploy will remove not only all tables for `branch1` and all of its associated areas, but the tables for `branch2` and all of its associated areas as well.

```
iw-home
└default
  └ main
    └branch1
      ├ STAGING
      ├ WA1
      ├ WA2
      └ branch2
        ├ STAGING
        ├ WA3
        └ WA4
```

1. Rename the following file:
   dd-home`/conf/synctracker.cfg.example`

   to:
   dd-home`/conf/synctracker.cfg`.

2. Then configure the appropriate database sections.

3. Execute the following command for *each area* in the directory structure to update the DataDeploy tracker table. You only need to perform this step once after DataDeploy is installed.

   ```
   iwsyncdb.ipl -synctracker vpath
   ```

   For the example shown above, you would execute the following commands:

   ```
   iwsyncdb.ipl -synctracker default/main/branch1
   iwsyncdb.ipl -synctracker default/main/branch1/STAGING
   iwsyncdb.ipl -synctracker default/main/branch1/WA1
   iwsyncdb.ipl -synctracker default/main/branch1/WA2
   iwsyncdb.ipl -synctracker default/main/branch1/branch2
   iwsyncdb.ipl -synctracker default/main/branch1/branch2/STAGING
   iwsyncdb.ipl -synctracker default/main/branch1/branch2/WA3
   iwsyncdb.ipl -synctracker default/main/branch1/branch2/WA4
   ```

See "iwsyncdb.ipl Usage" on page 189 for details about `iwsyncdb.ipl`.

Uninstalling DataDeploy

Peform the steps that correspond to your platform:

- (Solaris) Go to the *dd-home*`/install` directory and run `iwuninstalldd`.
- (Windows) Select DataDeploy from the Add/Remove Programs dialog box and click **Remove**.

Chapter 3

Deployment Concepts

This chapter describes the following general deployment concepts and components:

- How different methods of invoking DataDeploy affect your configuration activities.
- The roles and components of DataDeploy configuration files.
- What happens during a TeamSite-to-database deployment.

Among other concepts, the next chapter discusses tuples and data organization. It is recommended that you understand the concepts in this and the next chapter prior to configuring DataDeploy.

Ways to Invoke Deployment

There are several ways in which to invoke DataDeploy:

- From the command line.
- From an `iwat` trigger script.
- As a TeamSite workflow task that is not associated with TeamSite Templating.
- From the TeamSite Templating graphical user interface (GUI) using automated deployment, known as DAS.

Many of the examples in this book are related to DAS. See Chapter 8, "Automating Deployment with DAS," for full details.

All of these methods require the existence of one or more DataDeploy configuration files. The first three methods require that you manually create these configuration files. The last method creates all the necessary DataDeploy configuration files automatically after you have performed the necessary system setup. The following table shows a summary of each invocation method and its related tasks:

Invocation Method	Setup and Invocation Tasks	For More Information See...
Command Line	• Manually create a DataDeploy configuration file. • Execute `iwdd.ipl` from the command line. • Deployment occurs when you execute the command from the command line.	"Configuration File Details and Examples" on page 105; "iwdd.ipl Command" on page 159.
`iwat` Trigger Script	• Manually create a DataDeploy configuration file. • Create an iwat trigger script containing an `iwdd.ipl` command that references the DataDeploy configuration file. • Deployment occurs when the iwat script is triggered.	"Configuration File Details and Examples" on page 105; "iwdd.ipl Command" on page 159; *TeamSite Command Line Tools*.
Workflow Task	• Manually create a DataDeploy configuration file. • Create a workflow external task containing an iwdd.ipl command that references the DataDeploy configuration file. • Deployment occurs when the external task is executed.	"Configuration File Details and Examples" on page 105; "iwdd.ipl Command" on page 159; "Configuring TeamSite Workflow" in the *TeamSite Administration Guide*.
TeamSite Templating GUI	• Install TeamSite Templating. • Configure Database Auto-Synchronization (DAS). • Deployment occurs automatically whenever an end user modifies a data content record (DCR) through the TeamSite Templating GUI.	"Automating Deployment with DAS" on page 171; the *TeamSite Templating Developer's Guide*.

Configuration Files

DataDeploy configuration files let you specify the following:

- What, where, and how data is deployed.
- Whether DataDeploy will run as a client or server.

A TeamSite/DataDeploy installation can contain any number of configuration files. The most common scenario is for a system to contain multiple configuration files, one for each specific type of deployment.

For the "TeamSite Templating GUI" scenario shown in the preceding table, a DataDeploy configuration file is automatically created for each data type in the TeamSite Templating directory structure (in this context, *data type* refers to a directory in TeamSite Templating's directory structure; see page 50 or the *TeamSite Templating Developer's Guide* for more information about TeamSite Templating directories). The correct configuration file is then referenced automatically whenever a user changes a data content record for a given data type. For more information about automatic deployment, known as DAS, see Chapter 8, "Automating Deployment with DAS."

For the other scenarios shown in the preceding table, you must create each configuration file manually, and then name the file through a command line option for the `iwdd.ipl` command.

All configuration files, whether created manually or by DAS, contain the same file components. These components are described in the following subsection.

Note: If you are using DataDeploy in a non-US English environment, see Appendix D, "Internationalization."

File Components

All DataDeploy configuration files:

- Can have any name.
- Are in XML format.
- Reside by default in the *dd-home*`/conf` directory.

A configuration file is structured as a hierarchy of sections, each letting you control a different deployment parameter. A file can have any number of sections.

Parameters that you can set are:

- Filters that include and exclude possible data sources.
- Substitution rules to replace text and data values automatically during deployment.
- Client-specific parameters and activities.
- Type of deployment (TeamSite-to-database, XML-to-database, and so on).
- Source of extended attribute data (TeamSite, a database, or an XML file).
- Destination of extended attribute data (a database or an XML file).
- Details about source and destination data (specific fields to select, type of table to update or create, and so on).
- SQL commands that execute automatically during deployment.
- Server-specific parameters (for three-tier systems).

See the sample configuration file sections starting on page 109 for details about configuration file structure and syntax.

See “Invoking DataDeploy” on page 159 for more information about configuring DataDeploy as a client or server. See “Configuration File Details and Examples” on page 105 for more information about controlling all other DataDeploy parameters.

Incremental Deployment

DataDeploy can perform incremental deployments, in which it calculates the differences between any two specified vpaths and produces a delta table of the changes. The vpaths can be any two arbitrary TeamSite paths such as edition paths, staging area paths, or workarea paths. See Item 6, “Source Type” in “Sample File Notes” starting on page 115 for information about configuring an incremental deployment.

Deployment Scenarios

This section describes what happens when you execute a TeamSite-to-database deployment. This type of deployment is used as an example because it is the most commonly configured deployment type, it requires the most complex configuration files, and it is a type of deployment that you can perform manually (with the command line) or automatically, with DAS.

This section provides only an overview. To fully understand this section, including the discussion about base and delta tables, you should also study:

- Chapter 8, "Automating Deployment with DAS."
- "Sample TeamSite-to-Database Configuration File" on page 109 (describes manually configuring a file for a TeamSite-to-database deployment).

Other deployment scenarios such as TeamSite-to-XML, XML-to-XML, and so on, are essentially variations of the TeamSite-to-database deployment. These scenarios are described briefly starting on page 141.

Deploying from TeamSite to a Database: Overview

Whenever a TeamSite-to-database deployment finishes executing, the end result is an updated table on the destination system. This table will be either a *base* table, *delta* table, or *standalone* table, depending on what type of update you instruct DataDeploy to perform (as defined in the configuration file's `<update>` section). Update types are named for the type of table they modify. For example, a delta update modifies a delta table, and so on.

Details about each type are as follows:

- Base update: Extended attribute data is extracted from a TeamSite workarea, staging area, or edition, and is deployed to a base table containing full (as opposed to delta) data about the extended attributes. The most common sources of data for a base table are staging areas and editions. Whenever a base table is generated, an entry for that table is recorded in a tracker table residing in the database. See "Data Synchronization" on page 33 for more information.

- Delta update: On the TeamSite server, extended attribute data from a workarea is compared with the extended attribute data in a staging area or edition. Differences—the delta data—are identified and deployed to a delta table on the destination system. This table contains only the delta data from the workarea; it does not contain full static data about every item in the workarea (the delta table's associated base table should exist from a previous deployment). The relationship between the workarea data and the data in its parent area (a staging area or edition) is updated in the tracker table residing in the database. See "Data Synchronization" on page 33 for more information.
- Standalone update: Data is extracted from a TeamSite workarea, staging area, or edition and is deployed to a standalone table containing full data about the extended attributes. A standalone update differs from a base update in that it does not generate an entry in the tracker table.

Note: You can execute base and delta table updates manually (from the command line) or as part of automated deployment, known as DAS. When DAS is configured, certain TeamSite events automatically trigger deployment.

You can only execute standalone table updates manually; you cannot execute them with DAS.

Data Sources

When you deploy extended attribute data from TeamSite to a database, you can specify that it come from a TeamSite workarea, staging area, or edition. Of these three, workarea data is the only type that can be deployed using any of the three types of update (base, delta, or standalone). When deploying staging area or edition data, use a base update if you plan subsequent delta table generation, or use a standalone update if you do not need to track the table's relationship to other tables. The following table shows which data sources are supported for each type of update:

		Update Type		
		Base	**Delta**	**Standalone**
TeamSite Source Area	**Workarea**	Supported	Supported	Supported
	Staging Area	Supported	Not Supported	Supported
	Edition	Supported	Not Supported	Supported

Supported TeamSite Source Areas for Different Types of Update

Data Destinations

When you begin deployment, DataDeploy extracts information from the source that you specify and organizes this information internally as *tuples*.Tuples can then be deployed into a specified destination using selection and formatting rules defined in your DataDeploy configuration file(s).

All TeamSite tuples contain the following metadata:

- Exactly one *path* element, which is the area relative path name of the file associated with the tuple's key-value pair(s).
- One or more *key-value pairs*. The key is the name (also known as the *class*) of the extended attribute. For example, `News-Section` is the key of the extended attribute `News-Section:Sports`. The value is the data value for a tuple's key—`Sports`, in this example.
- Exactly one *state* element, which describes the status of the tuple. Possible values are `Original`, `New`, `Modified`, and `NotPresent`. See "Data Destinations" on page 29 for details about the state element's values.

Tuples are deployed in different ways, depending on the organizational structure, or schema. DataDeploy deploys data to databases using three different organizational structures:

- Narrow tuples (which result in narrow or wide tables when the tuples are deployed to a database; any given record is mapped to a single narrow or wide table)
- Wide tuples (which result in wide tables when the tuples are deployed to a database; any given record is mapped to a single wide table)
- User-defined database schemas (which result in multiple tables when deployed to a database; any given data content record may be mapped to multiple tables)

For more information on tuples and the three different types, see Chapter 3, "Deployment Concepts."

In a TeamSite-to-database deployment, the destination of data can be any stored in any database on a DataDeploy server (in a three-tier system) or a database on an application server (on a two-tier system).

When the base table (representing Staging) is initially populated, all tuples (entries) will have the state of `Original`. Over the life of the base table, after submits, new tuples are added in the table and these tuples will have the state of `New`. If a particular tuple in a workarea is changed and submitted, and if that tuple already exists in the base table, the tuple will have a state of `Modified`. See "Updating a Base Table" on page 36 for an example and more details.

The tuples in standalone tables do not have a state.

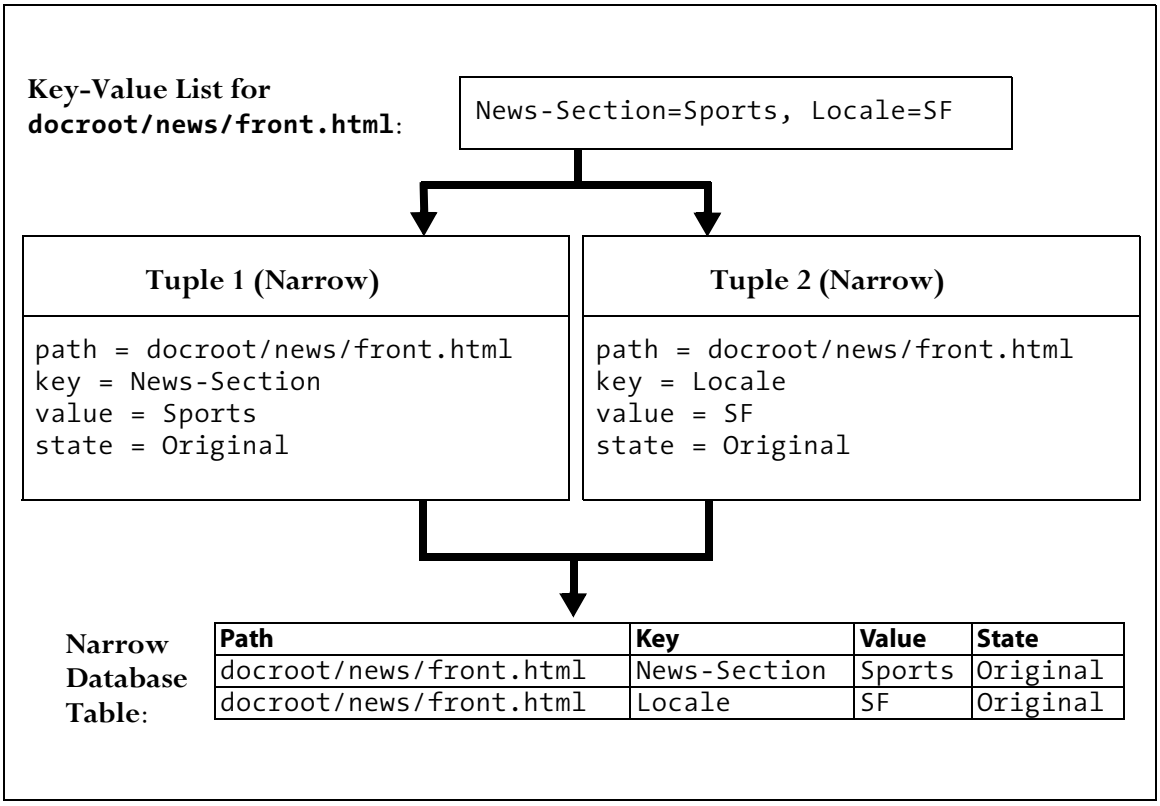
In a delta table, the state is the tuple's status as viewed in TeamSite area 1 (most often the staging area) relative to the tuple in TeamSite area 2 (most often a workarea). A delta table can have tuples states of `Original`, `New`, `Modified`, or `NotPresent`. The following table shows the scenarios that can cause these states:

A delta table tuple state of:	**Was caused by:**
`Original`	Merging delta data from another workarea into a base table through a base update (such as when submitting the other workarea data to a staging area).
`New`	Generating a new tuple through a delta update (such as when adding a new extended attribute to a file in a workarea).
`Modified`	Updating a delta table through a delta update.
`NotPresent`	Data existing in a base area but not in a workarea (such as when the data is deleted from the workarea, or when data is newly added to the base area from a different workarea).

Delta Table Tuple States

Base Table Format: Narrow Tuples

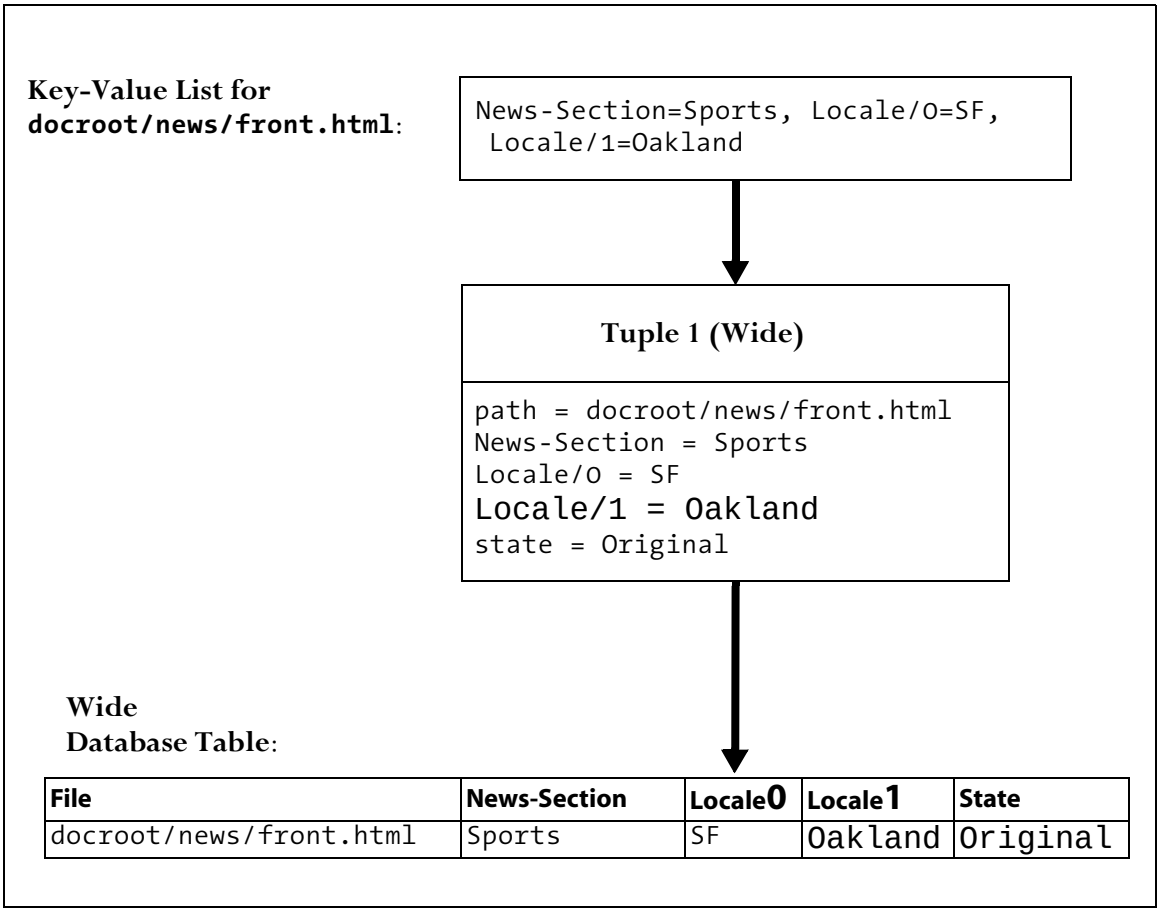
By default, deploying narrow tuples creates a base table in a database containing columns for Path, Key, Value, and State. For example:



Narrow Tuple Default Base Table

Base Table Format: Wide Tuples

By default, wide tuples deploy into wide tables, in which key values from the tuple are placed in separate columns. The end result is a table in which a single file record contains individual key value columns. For example:



Wide Tuple Default Base Table

It is also possible to deploy narrow tuples into a wide table by configuring DataDeploy to use wide tuples. When you do, the tuples are deployed to a wide table by default. See "Sample TeamSite-to-Database Configuration File" on page 109 for guidelines about specifying wide versus narrow tuples.

You can also deploy narrow tuples to a wide table by manually configuring a set of SQL commands in the DataDeploy configuration file. These SQL commands would then execute automatically during deployment. Detailed SQL commands are beyond the scope of this document; you should refer to third party SQL documentation for more information about that topic.

Note: Table column names cannot contain reserved SQL characters such as dash (-), slash (/), question mark (?), percent (%), and so forth.

Data Synchronization

On the database system, DataDeploy must keep a record of which delta tables are associated with which base tables (assuming you are using automated deployment, known as DAS. This is necessary so that delta tables from multiple workareas that are associated with a single base table from a staging area will remain synchronized when changes from one workarea are submitted to the staging area. This relationship is maintained by the tracker table residing in the same database as the base and delta tables.

Deploying from TeamSite to a Database: Details

The most common sequence of events when deploying from TeamSite to a database is as follows:

1. Generating an initial base table of a staging area or edition.
2. Generating a delta table for each workarea associated with the staging area or edition from Item 1.
3. Configuring TeamSite to invoke DataDeploy so that the base table from Item 1 is automatically updated whenever changes are about to be submitted to its corresponding staging area or edition from a workarea.

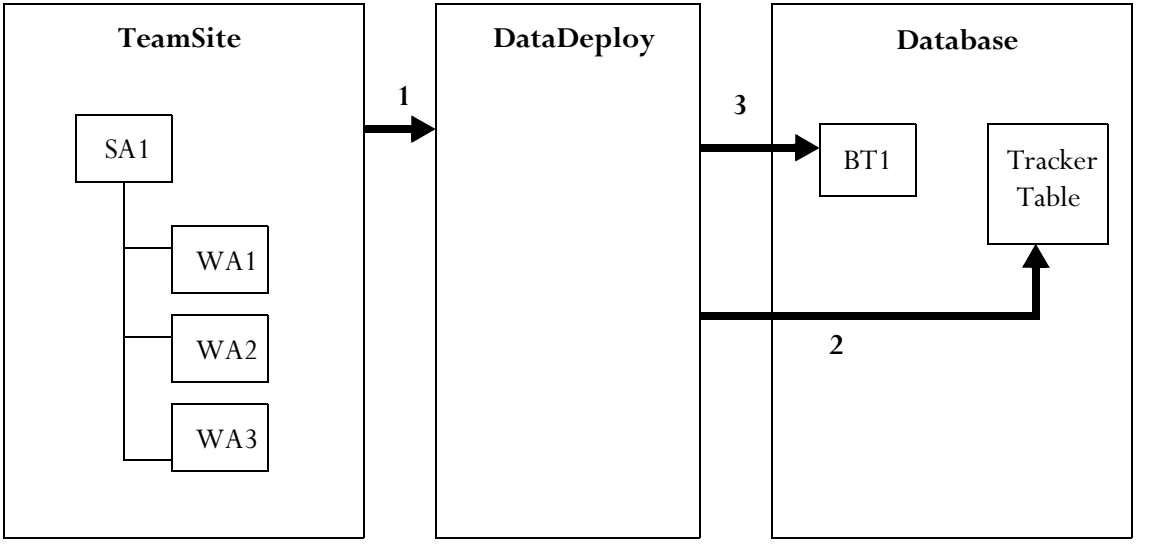
This subsection describes these steps, which are part of automated deployment, known as DAS.

Generating an Initial Base Table

Usually, the first action you will instruct DataDeploy to perform is the creation of an initial base table for a staging area or an edition. The following example shows the creation of a base table BT1 from a staging area SA1 on a TeamSite branch such as `/default/main/branch1`. The configuration file for this deployment is shown in "Starting-State Base Table Configuration File" on page 155.

Note: In that file, the value of the attribute `name` in the `path` element is relative to the staging area that is the source of the data being deployed. Based on the preceding conditions, the following sequence of events occurs. Refer to the figure following this list for a keyed diagram of the steps.

1. Invoke DataDeploy from the command line, specifying the deployment configuration file that contains the preceding parameters. DataDeploy reads the configuration file and goes to SA1, extracting all extended attribute data.
2. DataDeploy creates the Tracker Table (or updates it if it already exists) to track relationships between base and delta tables.
3. Based on additional information in the configuration file, DataDeploy creates base table BT1 in the destination database, populating it with the data from Step 1.

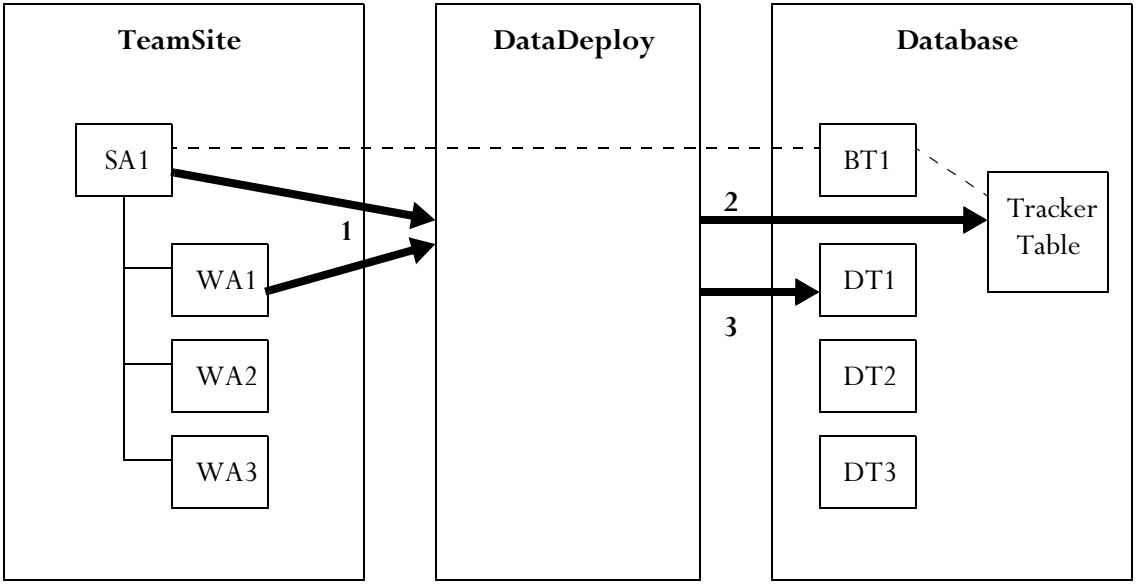


Generating an Initial Base Table

Generating a Delta Table

After creating the initial base table, you will need to generate one or more delta tables based on the workareas associated with the base table's staging area. This example shows the creation of a delta table DT1 from a workarea WA1. It assumes that a base table for SA1 has already been generated, and that WA1 is a workarea of staging area SA1. Based on the preceding conditions, the following sequence of events occurs. Refer to the figure following this list for a keyed diagram of the steps.

1. Invoke DataDeploy from the command line, specifying the deployment configuration file that contains the preceding parameters. DataDeploy compares the extended attribute data in WA1 with the same data in SA1 to determine the tuple difference between the two areas.
2. DataDeploy updates the Tracker Table to record that DT1 is a child of BT1.
3. DataDeploy creates DT1, using the delta data it determined in Step 1. If there is no delta data, it creates an empty delta table.



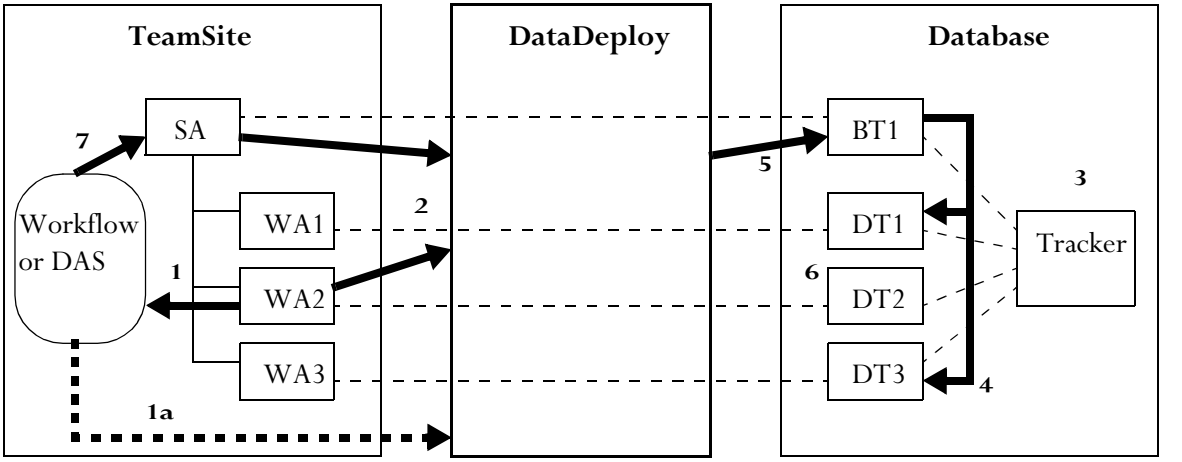
Generating a Delta Table

Updating a Base Table

After creating the initial base and delta tables, you can configure TeamSite workflow to automatically update a base table whenever changes in a workarea are about to be submitted to a staging area. This example assumes the following:

- You plan to submit a file list (rather than the entire workarea) from workarea WA2 to a staging area SA1.
- A base table BT1 already exists for staging area SA1.
- Delta tables DT1 through DT3 already exist for all workareas (WA1 through WA3) associated with staging area SA1.
- A tracker table already exists to establish and track the relationships between the base and delta tables.

Based on the preceding conditions, the following sequence of events occurs. Note that all of the DataDeploy activity takes place before TeamSite submits the changes from WA2. Refer to the figure following this list for a keyed diagram of the step.



Updating a Base Table

1. If the submission occurs as part of a TeamSite workflow job, the TeamSite workflow engine obtains a list of files to be submitted from WA2 to SA1. If Database Auto-Synchronization (DAS) is configured, as described in Chapter 8, "Automating Deployment with DAS," DAS obtains the list of files to be submitted. This list of files is then passed to DataDeploy (1a in the following figure).
2. DataDeploy compares the file list items in WA2 with the same items in SA1 to determine the tuple differences between the two areas. These differences will be installed in BT1 in Step 5.
3. DataDeploy checks the tracker table to determine the children of BT1.
4. Original rows from BT1 are propagated to DT1 and DT3 (but not to DT2). This ensures that the original rows in BT1 are not lost, but instead are stored as now-obsolete data in its child delta tables.
5. DataDeploy updates BT1 with the data derived earlier in Step 2.
6. DataDeploy removes from DT2 all rows whose path and key values are identical to those being submitted from WA2 to SA1. This ensures that items not being submitted from WA2 to SA1 are retained in DT2.
7. The workflow engine completes the submission of the file list to SA1.

Table Updates

Hypothetical table updates for a scenario fitting this model would proceed as follows. For simplicity, the tables shown here have column headings identical to the tuple items Path, Key, Value, and State. In most situations, the columns will have other names. Because the term "key" has a specific meaning in many database languages, it is recommended that you do not use "key" as a column heading.

Starting State[1]

BT1			
Path	**Key**	**Value**	**State**
P1	K1	V1	Orig

DT1			
Path	**Key**	**Value**	**State**

DT2			
Path	**Key**	**Value**	**State**

Event 1[2]

BT1			
Path	**Key**	**Value**	**State**
P1	K1	V1	Orig

DT1			
Path	**Key**	**Value**	**State**

DT2			
Path	**Key**	**Value**	**State**
P2	K2	V2	New

Event 2[3]

BT1			
Path	**Key**	**Value**	**State**
P1	K1	V1	Orig
P2	K2	V2	Orig

DT1			
Path	**Key**	**Value**	**State**
P2	K2	V2	NtPres

DT2			
Path	**Key**	**Value**	**State**

Sample Table Updates

1. In their starting state, all tables are synchronized. Because there are no differences between SA1, WA1, and WA2, there is no delta data. Therefore, DT1 and DT2 are empty. This is the starting state that would exist if you completed the steps described in "Generating an Initial Base Table" on page 34. The configuration file for generating this initial version of BT1 is shown in "Starting-State Base Table Configuration File" on page 155.

2. In Event 1, workarea WA2 is changed locally with new data P2, K2, and V2, but the changes are not submitted to staging area SA1. Because the changes are not submitted, you must execute a delta update so that delta table DT2 reflects the new data in WA2. During this delta update, DataDeploy identifies the differences between SA1 and WA2 and records these differences (the delta data) in DT2. This scenario is similar to the scenario in "Generating a Delta Table" on page 35.

However, in that scenario a delta table did not exist yet and had to be generated for the first time. In the scenario shown here, the delta tables already exist and therefore only need to be updated. The configuration file for this delta deployment is shown in "Event 1 Configuration File" on page 156.

3. In Event 2, workarea WA2 (complete with its changes from Event 1) is submitted to staging area SA1. In the configuration file for this deployment, Path and Key were named as the basis-for-comparison columns. Therefore, DataDeploy compares the Event 1 values of these columns in BT1 and DT2, sees that they are different, and determines that the row from DT2 Event 1 should append rather than replace the data in BT1. DT1 has the new values shown here because WA1 now differs from SA1. If necessary, a Get Latest operation in WA1 would bring WA1 into sync with SA1. (Had Event 1 DT2 contained a P1 K1 V2 row, it would have replaced rather than appended the original BT1 row. In that case, the original BT1 row would have been propagated to DT1, after which P1 K1 V2 would have replaced P1 K1 V1 in BT1. A subsequent Get Latest in WA1 would bring WA1 into sync with SA1, and the data in DT1 would be deleted). DT2 is empty because WA1 is once again in sync with SA1. This is the ending state that would exist if you completed the steps described in "Updating a Base Table" on page 36. The configuration file for this deployment is shown in "Event 2 Configuration File" on page 157. Note: In that file, all of the items in `filelist` are path-relative to `area`.

Composite Table Views

There are three ways that you can create table views:

- Through SQL commands that you execute manually to query the database after it is created. See "Querying Tables" on page 197 for more information.
- Through SQL commands named in the `user-action` attribute of the DataDeploy configuration file's `<sql>` element. You run these commands by executing an SQL-specific deployment that you specify through the command line options `iwdd-op=do-sql` and `user-op=`*`anyname`*. See "Sample File Notes" on page 115 and "Invoking DataDeploy" on page 159 for more information.
- By setting the `table-view` attribute in the DataDeploy configuration file's `<database>` section. See "Sample File Notes" on page 115 for more information.

The following composite views for workareas WA1 and WA2 would result from the scenarios described in the previous sections. The composite for WA1 is the result of querying BT1 and DT1 using the SQL statements described in "Querying Tables" on page 197. Likewise, the composite for WA2 is the result of querying BT1 and DT2.

Starting State

WA1			
Path	**Key**	**Value**	**State**
P1	K1	V1	Orig

WA2			
Path	**Key**	**Value**	**State**
P1	K1	V1	Orig

Event 1

WA1			
Path	**Key**	**Value**	**State**
P1	K1	V1	Orig

WA2			
Path	**Key**	**Value**	**State**
P1	K1	V1	Orig
P2	K2	V2	New

Event 2

WA1			
Path	**Key**	**Value**	**State**
P1	K1	V1	Orig

WA2			
Path	**Key**	**Value**	**State**
P1	K1	V1	Orig
P2	K2	V2	Orig

Composite Table Views

Chapter 4

Data Organization

This chapter contains the following sections:

- Overview
- Deploying Data with Narrow Tuples
- Deploying Data with Wide Tuples
- Deploying Data with User-Defined Database Schemas: Overview
- Deploying Data with User-Defined Database Schemas: Architectural Details
- iwsyncdb.ipl Support for User-Defined Database Schemas
- Deploying Data with User-Defined Database Schemas: Partial Support of Metadata Deployment
- Other Data Organization Issues

Overview

DataDeploy's main purpose is to transfer:

- File metadata (also called extended attributes).
- XML data.
- TeamSite Templating data content records (which are special XML files).

You can create file metadata by using:

- The `iwextattr` command line tool.
- The `File > Set Metadata` command in the TeamSite GUI (this menu item does not appear by default).
- A workflow instance, or job, as a part of a `<cgitask element>`.

For more information, see *TeamSite Command-Line Tools* and the *TeamSite Administration Guide*.

When you begin deployment, DataDeploy extracts information from the source that you specify and organizes this information internally as *tuples*.Tuples can then be deployed into a specified destination using selection and formatting rules defined in your DataDeploy configuration file(s).

All TeamSite tuples contain the following metadata:

- Exactly one *path* element, which is the area relative path name of the file associated with the tuple's key-value pair(s).
- One or more *key-value pairs*. The key is the name (also known as the *class*) of the extended attribute. For example, `News-Section` is the key of the extended attribute `News-Section:Sports`. The value is the data value for a tuple's key—`Sports`, in this example.
- Exactly one *state* element, which describes the status of the tuple. Possible values are `Original`, `New`, `Modified`, and `NotPresent`. See "Data Destinations" on page 29 for details about the state element's values.

Tuples are deployed in different ways, depending on the organizational structure, or schema. DataDeploy deploys data to databases using three different organizational structures:

- Narrow tuples (which result in narrow or wide tables when the tuples are deployed to a database; any given record is mapped to a single row in a narrow or wide table)
- Wide tuples (which result in wide tables when the tuples are deployed to a database; any given record is mapped to a single row in a wide table)
- User-defined database schemas (which result in multiple tables when deployed to a database; any given record may be mapped to rows in multiple tables)

The following sections describe these three organizational structures.

Note: The following sections refer to `iwsyncdb.ipl`, which is a command-line tool that creates DataDeploy configuration files by running a script called `ddgen.ipl`; these files determine how databases organize data received from DataDeploy.

`iwsyncdb.ipl` is used to setup automated deployment, known as DAS. For a complete list of the command options for this tool, see "iwsyncdb.ipl Usage" on page 189. To deploy data content records or extended attributes automatically, by causing TeamSite events to trigger the `ddgen.ipl` script, see Chapter 8, "Automating Deployment with DAS."

Much of this chapter describes the role of `iwsyncdb.ipl` in implementing user-defined database schemas.

Deploying Data with Narrow Tuples

Narrow tuples are not a common way to deploy data due to structural constraints. Narrow tuples contain exactly one path, key, value, and state element. For example, the following figures show DataDeploy's internal representation of two narrow tuples. Tuple 1 is for the `News-Section:Sports` extended attribute for the file `docroot/news/front.html`. Tuple 2 is for the `Locale:SF` extended attribute for the same file. Note that because a narrow tuple can contain only one key-value pair, DataDeploy must create multiple tuples (two in this case) if a file's extended attributes consist of more than one key-value pair.

Tuple 1

`path = docroot/news/front.html`
`key = News-Section`
`value = Sports`
`state = Original`

Tuple 2

`path = docroot/news/front.html`
`key = Locale`
`value = SF`
`state = Original`

Narrow Tuples

Note: You cannot deploy data content records using narrow tuples.

To *manually* deploy extended attributes with narrow tuples to a database:

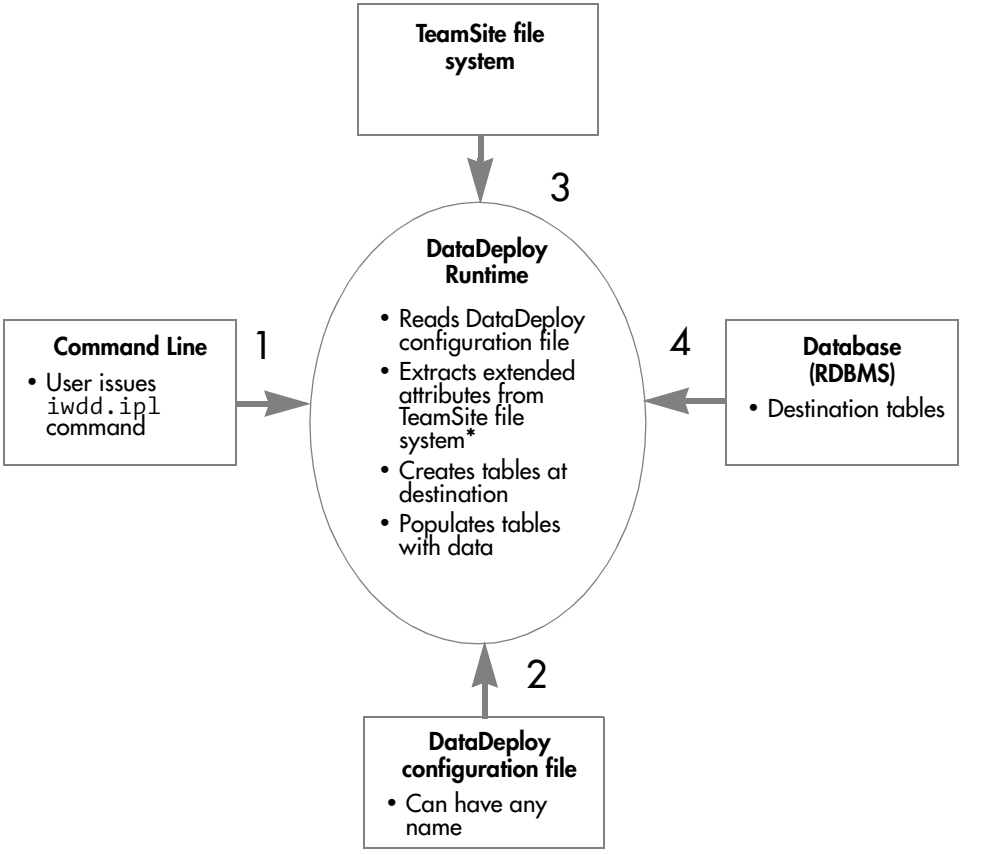
1. Create and edit a DataDeploy configuration file; this file can have any name.

   For information on creating a DataDeploy configuration file for TeamSite-to-database deployment, see "Sample TeamSite-to-Database Configuration File" on page 109.

2. Run `iwdd.ipl` to deploy your data.

   For information on `iwdd.ipl`, see Chapter 6, "Invoking DataDeploy."

The following figure illustrates this process:



Manual Deployment Process to a Database

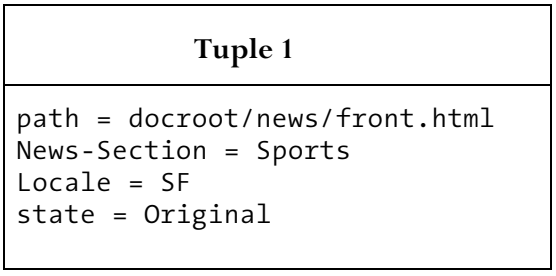
* Note: If you were deploying a data content record, DataDeploy would extract the data content record from the TeamSite file system.

You cannot automatically deploy narrow tuples with DAS. DAS only accepts wide tuples. See Chapter 8, "Automating Deployment with DAS," for details about automating deployment.

Deploying Data with Wide Tuples

You can deploy extended attributes, XML data, or data content records as wide tuples. If these tuples are then sent to a database, the database will store them in wide tables. If the destination is an XML file, the tuples are stored as tags and values.

Data content records must be deployed using either wide tuples or user-defined database schemas. By default, wide tuples deploy into wide tables when sent to a database. Wide tuples contain exactly one path element and one state element, and any number of key-value pairs. Thus, a file's extended attribute data can be represented in a single wide tuple even if the extended attributes consist of more than one key-value pair. The following figure shows DataDeploy's internal representation of a wide tuple. The information shown here is the same as that shown in the previous example. The only difference is that in this case, DataDeploy was configured to create a wide tuple.



Wide Tuple

In a wide tuple, DataDeploy eliminates the `key =` and `value =` labels for the key and value data, instead using the format *`key = value`* for each key-value pair. This arrangement simplifies the creation of a wide base table as described in "Base Table Format: Wide Tuples" on page 32.

Support for wide tuples requires that all extended attribute keys be unique. For example, a file cannot have two keys named `Locale`. To satisfy this requirement, TeamSite uses a numeric suffix for key names that would otherwise be unique. For example, if the file `docroot/news/front.html` has two `Locale` keys with the values `SF` and `Oakland`, the keys are named `Locale/0` and `Locale/1`.

The TeamSite GUI and metadata capture module automatically enforce this naming convention when you create extended attributes for a file. The resulting wide tuple in this example is as follows:

Tuple 1
path = docroot/news/front.html News-Section = Sports Locale/0 = SF Locale/1 = Oakland state = Original

Wide Tuple with Similar Locale Keys

To *manually* deploy extended attributes with narrow tuples to a database:

1. Create and edit a DataDeploy configuration file; this file can have any name.

   For information on creating a DataDeploy configuration file for TeamSite-to-database deployment, see "Sample TeamSite-to-Database Configuration File" on page 109.

2. Run `iwdd.ipl` to deploy your data.

   For information on `iwdd.ipl`, see Chapter 6, "Invoking DataDeploy."

For an illustration of this process, see the figure on page 44.

The remainder of this section discusses deploying data content records automatically using DAS. The discussion in this section is only an overview; its purpose is to provide a conceptual understanding of automated wide tuple deployment of data content records so that you can contrast this method with automated deployment of data content records with user-defined database schemas. For more information about DAS, see Chapter 8, "Automating Deployment with DAS."

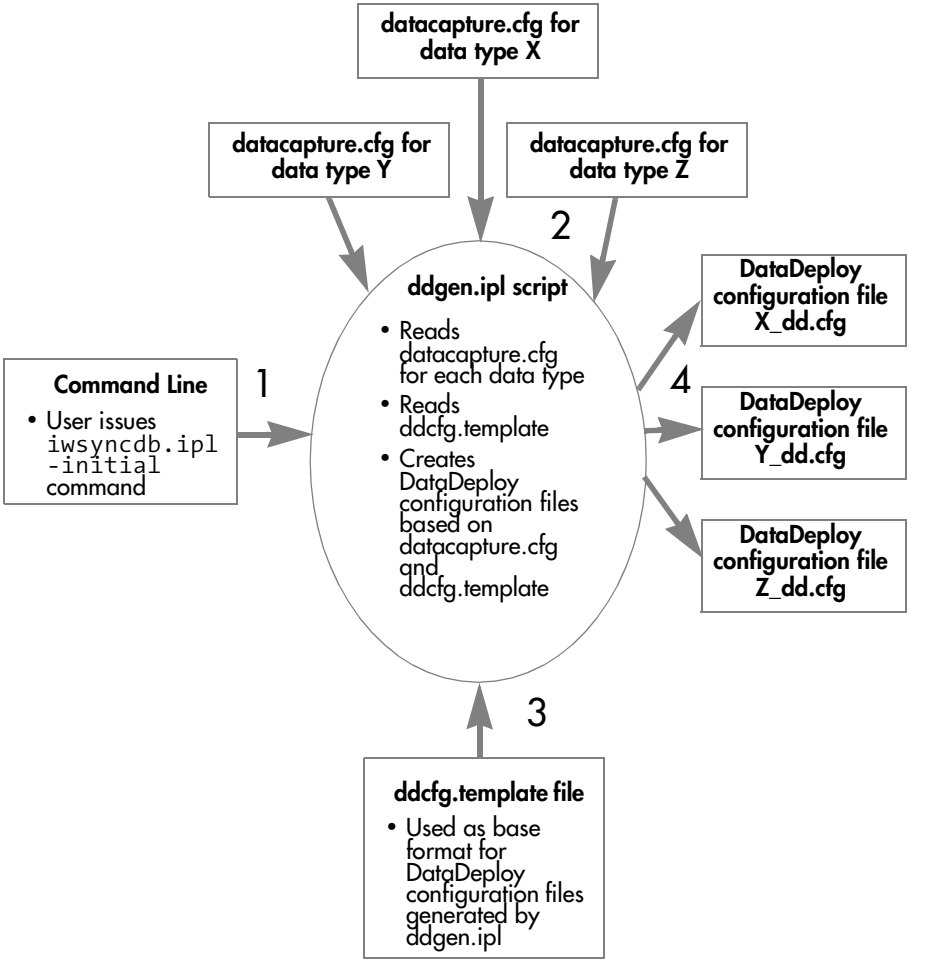
When using DAS, administrators create table structures by using the `iwsyncdb.ipl` command line tool (`iwsyncdb.ipl -initial` or `iwsyncdb.ipl -ddgen`). When you specify a wide or narrow table architecture, this command line tool creates table structures in the following way:

- `iwsyncdb.ipl` reads the data capture template, `datacapture.cfg`, that corresponds to each data type in a workarea's vpath.
- `iwsyncdb.ipl` reads the DataDeploy template configuration file, `ddcfg.template`, which is located in the following path: *`dd-home`*`/conf/ddcfg.template`.
- `iwsyncdb.ipl`, as a result of this input, creates a DataDeploy configuration file for each data type; this file is named *`data_type`*`_dd.cfg`, where `data_`*`type`* is a variable that represents the data type's name.
- `iwsyncdb.ipl` generates a single table for the entire data capture template and maps each field in the data content record to a single table column.

Note: In this context, *data type* refers to a directory in TeamSite Templating's directory structure; see page 50.

For replicant fields (a field that contains a repeating group of values), `iwsyncdb.ipl` creates *`n`* number of columns, where *`n`* is a variable that represents the maximum number of occurrences for a given replicant in a data type's data capture template. When a data capture template contains many replicant fields, user-defined database schemas may offer a better procedure for deploying your data.

The following figure represents the architectural model for using DAS to create DataDeploy configuration files for data content records using wide tuples:



Deployment Using Wide or Narrow Tuples

The following sections describe the architectural model for the deployment of data content records to user-defined database schemas. When using DAS, DataDeploy will use the model in the previous figure if you choose not to use user-defined database schemas.

Deploying Data to User-Defined Database Schemas: Overview

A *schema* is the organization or structure for a relational database, including the layout of tables and columns. The primary objective of DataDeploy is to allow users to map their data content records to multiple user-defined database schemas. This feature is backward compatible with earlier versions of DataDeploy.

Prior to the 5.0 release, DataDeploy deployed a data content record to a single row in a database table. This deployment mapped every field (including replicant values) in a data content record to a column. This architecture is known as wide table mapping. DataDeploy will continue to support this approach; you can even use wide table mapping on some TeamSite branches and user-defined database schemas on others.

However, user-defined database schemas give greater flexibility to users who require that database content be distributed in normalized tables with relationships across tables. User-defined database schemas allow you to map a given record to rows in multiple tables that you can define with foreign and primary keys. The resultant tables have normalized data schemas. Normalized schemas make it easy to understand the relationships between tables. Tables with normalized schemas are also easy to query.

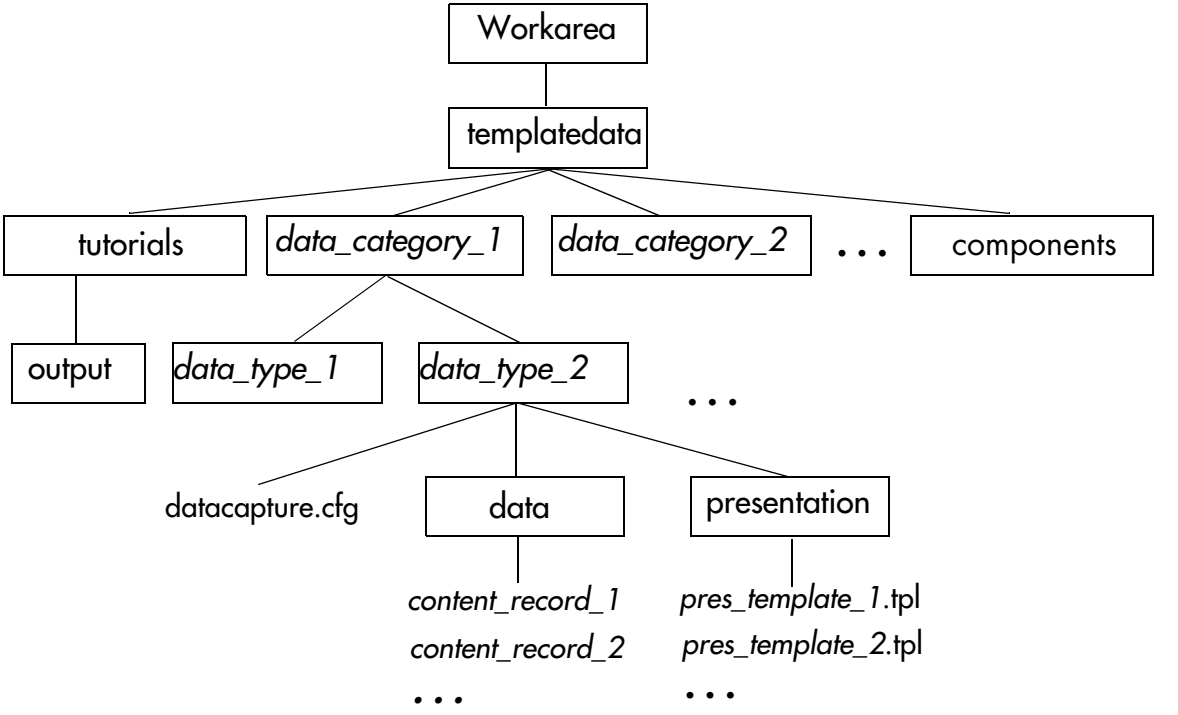
Users who wish to deploy data content records with many replicant fields (as described in this section) may wish to use user-defined database schemas instead of wide table mapping. User-defined database schemas offer more flexibility because you can map a single data content record to multiple tables, avoiding problems related to physical limitations of databases.

Note: Data content records are part of TeamSite Templating. To deploy data content records using user-defined database schemas, these records must be based on the Interwoven DTD or on customized data capture template DTDs (DTDs specify the rules' framework for creating data capture templates). These files generate the graphical user interface (GUI), or form, that allows users to create data content records.

The remainder of this chapter focuses on implementing user-defined database schemas for deployment of data content records. However, DataDeploy offers partial support for deploying TeamSite extended attributes (metadata). See “Deploying Data to User-Defined Database Schemas: Support for Metadata Deployment” on page 80.

It is possible to use user-defined database schemas manually. However, the examples in this section assume you are deploying automatically, with DAS. For more information on DAS, see Chapter 8, "Automating Deployment with DAS."

A specific example will clarify the need for mapping to multiple tables. TeamSite Templating uses a data storage hierarchy based on directories that contain data categories and types, as shown in the following figure:



TeamSite Templating Directory Structure

Note: Data types are analogous to subcategories. For more information, see the *TeamSite Templating Developer's Guide*.

Data categories contain one or more datadata types. Suppose that Pat, an administrator at XYZ Corporation, has used this hierarchy to create the following categories: `Internet` and `Intranet`. Also suppose that the `Internet` category has seven types: `auction`, `book`, `careers`, `medical`, `periodic`, `pr`, and `yacht`.

Suppose that Pat has also configured the following:

- She has created a data capture template—an XML file called `datacapture.cfg`—and has inserted it into the `book` directory. (Each data type must have its own data capture template.)
- She has configured this data capture template so that it will generate a form containing various fields that a user must complete or select.
- She has created a *replicant* element in the data capture template corresponding to the `book` type; this element will create a button in the data content form. Content Contributors must complete this form prior to submitting a data content record. In this example, Pat has used the replicant element to create a **Reviewers** button that a Content Contributor clicks each time he wishes to specify an additional reviewer.
- She has configured the data capture template so that a user can specify up to 10 reviewers.
- Each `reviewer` element has the following subelements: `Name`, `E-Mail`, and `Comments`.

Note: Pat would need to perform additional configuration to set up TeamSite Templating and DataDeploy, but that configuration is not relevant to this example.

Prior to the DataDeploy 5.0 release, DataDeploy used wide table mapping for deployment of data content records. Under that architecture, when an administrator created tables in a database using DAS (by using the command-line tool `iwsyncdb.ipl -initial` or `iwsyncdb.ipl -ddgen`), the table headings in the destination relational database connected to DataDeploy would have looked like this:

Path	Author	ISBN	Publisher	Title	Name0...	E-Mail0...	Comments0...	State

The ellipses (…) in the column headings indicate that DataDeploy would create additional column headings for each replicant field, up to the maximum number of fields indicated in the data capture template. In this example, Pat indicated a maximum of 10 fields for the Reviewers replicant by giving the `max` attribute a value of 10.

Therefore, each Reviewers' subelement would contain 10 headings:

- `Name0-Name9`
- `E-Mail0-E-Mail9`
- `Comments0-Comments9`

Under the wide table architecture, when Content Contributors at XYZ Corporation submitted a data content record with replicant information, DataDeploy would have sent the data content record to a wide table in a database. For the first data content record, created by Chris, DataDeploy would have created the following columns:

Path	Author	ISBN	Publisher	Title	Name0...	E-Mail0...	Comments0...	State
*mypath*0	William Faulkner	1234-56	Software Inc.	Using Data	Chris	chris@xyz.com	This book describes…	original

Note: `datacapture.cfg` determines the column-heading names.

Assume another user at XYZ Corporation, Don, submitted a second data content record to the same data type, `book`. Assume also that he did not specify any reviewers. After he submitted the data content record, DataDeploy added a new row in the previous table. DataDeploy then inserted the Path information and the State information in the appropriate columns. DataDeploy also inserted information that Don typed or selected in the data content record, as in the following example:

Path	Author	ISBN	Publisher	Title	Name0...	E-Mail0...	Comments0...	State
*mypath*0	William Faulkner	1234-56	Software Inc.	Using Data	Chris	chris@xyz.com	This book describes…	original
*mypath*1	Harper Lee	1256-89	Software Inc.	Using Tuples				original

Such data structures, created by mapping data to wide tables, present the following challenges:

- Tables can become so wide that they violate database limits, causing deployments to fail.
- When a data content record is deployed into a database schema defined by a third-party application server, an administrator may need to use native database techniques (such as stored procedures or triggers) to transform data from the wide table model into the required schema.
- Some key-value pairs will have no values.
- Data stored in such tables is not normalized. If users at XYZ Corporation create additional data content records using the wide table architecture, administrators will only be able to assume that the Path column contains unique information.

As a result, DataDeploy enables users to map their data content records to multiple tables, using user-defined database schemas schemas. This architecture allows administrators to normalize data. The following tables show a normalized data structure for the previous example:

Path	Author	ISBN	Publisher	Title	State
*mypath*0	Faulkner	1234-56	Software Inc.	Using Data	original
*mypath*1	Harper Lee	1256-89	Software Inc.	Using Tuples	original

Name0...	E-Mail0...	Comments0...	ISBN
Chris	chris@soft.com	This book describes…	1234-56

Deploying Data to User-Defined Database Schemas: Architectural Details

This section describes:

- Creating database tables with user-defined database schemas.
- Examples of how to format new configuration files that implement user-defined database schemas.
- Guidelines on coding new configuration files.

Creating Database Tables with User-Defined Database Schemas

The following files are used to map DCRs to database schemas:

- `dbschema.cfg`
- `ddcfg_uds.template`

Note: You should not create these files if you will not use user-defined database schemas.

The `dbschema.cfg` file must be defined once for each data type. This file allows administrators to specify the mapping of fields from data content records into custom-defined groups of columns, using the `<groups>` element. These groups are analogous to tables, in database terminology, and are treated as tables by DataDeploy.

In addition to the `dbschema.cfg` file, the DataDeploy architecture uses a DataDeploy template configuration file, called `ddcfg_uds.template`, which defines a skeletal configuration for automating deployment through DAS in conjunction with user-defined database schemas.

In summary, `iwsyncdb.ipl -initial` or `iwsyncdb.ipl -ddgen` requires the following files files for a wide table deployment or for a deployment with user-defined database schemas:

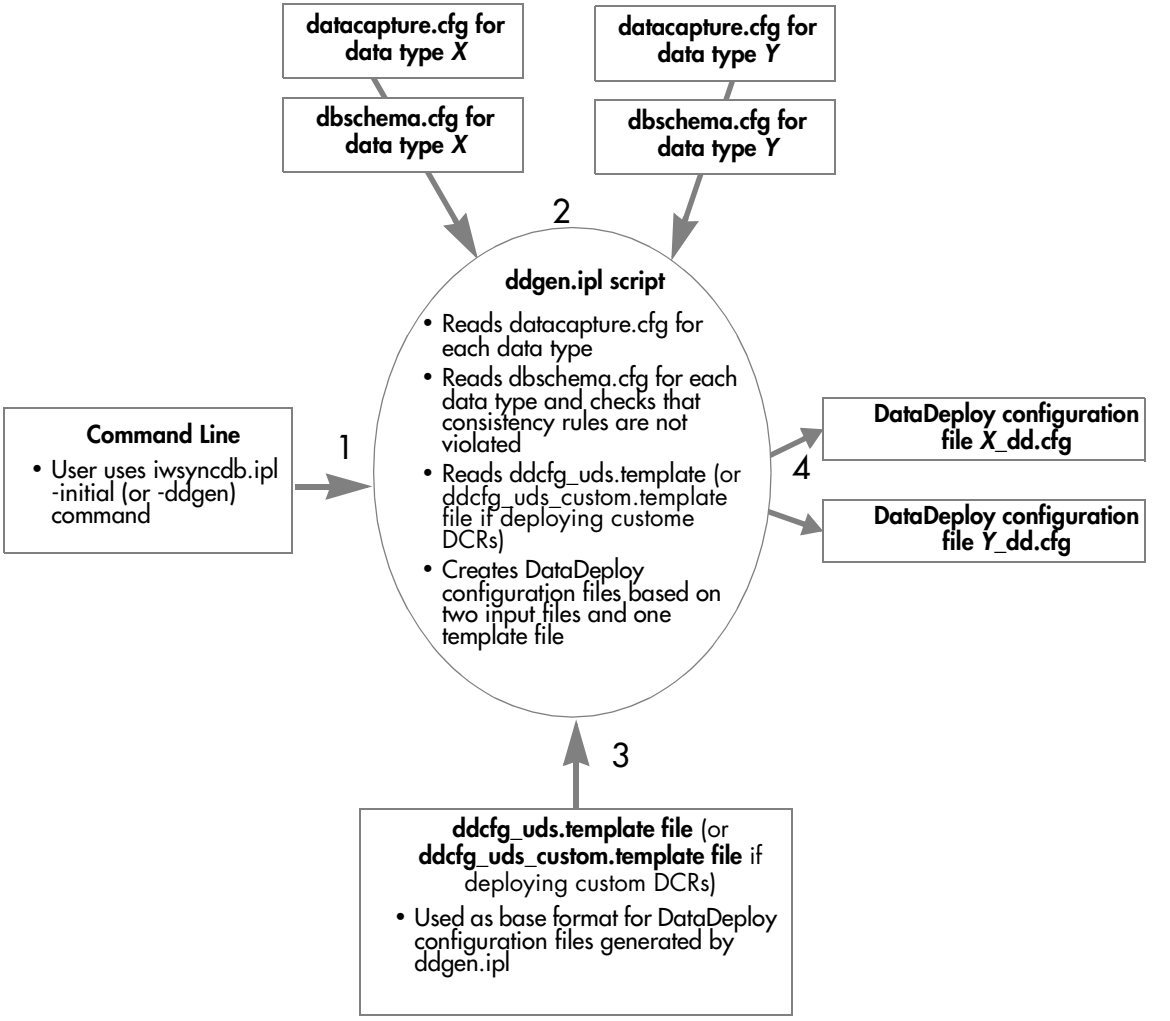
Deployment Using Wide Tuples	Deployment with User-Defined Database Schemas
area/templatedata/*data_category*/*data_type*/datacapture.cfg	*area*/templatedata/*data_category*/*data_type*/datacapture.cfg
dd-home/conf/ddcfg.template	*dd-home*/conf/ddcfg_uds.template or *dd-home*/conf/ddcfg_uds_custome.template if you are deploying custom DCRs.
	area/templatedata/*data_category*/*data_type*/dbschema.cfg

`iwsyncdb.ipl -initial` or `iwsyncdb -ddgen` will check for the presence of the required files and, depending on what files are present, will do one of the following:

- It will create a *data_type*_dd.cfg file that will result in the creation of wide tables. The path of this file will be: *area*/templatedata/*data_category*/*data_type*/*data_type*_dd.cfg.

- Or it will create a DataDeploy configuration file (*data_type*_dd.cfg) that will result in the creation of user-defined tables. The path of this file will be: *area*/templatedata/*data_category*/*data_type*/*data_type*_dd.cfg.
- Or it will return an error.

The following figure illustrates the deployment of data content records using the architecture for user-defined database schemas:



Deployment Process for Data Content Records Using User-Defined Database Schemas

Note: The primary difference between the basic DataDeploy configuration template file and DataDeploy configuration template files for user-defined schemas is that the ones for user-defined schemas do not use the `<select>` and `<update>` elements. Instead, they employ a more flexible syntax that has elements in the `dbschema.cfg` file that allow administrators to define primary keys and foreign keys, in addition to columns and tables.

The DTD of the `dbschema.cfg` file is as follows:

```
<!ELEMENT dbschema        (group)+ >
<!ELEMENT group               (attrmap, keys, create-sql?, exists-sql?) >
<!ATTLIST group name CDATA #REQUIRED
          root-group (yes|no) "yes"
          table CDATA #IMPLIED
          base-table CDATA #IMPLIED >
<!ELEMENT attrmap (column)+ >
<!ELEMENT column (EMPTY)>
<!ATTLIST column name CDATA        #REQUIRED
                 data-type CDATA        #REQUIRED
                 value-from-field CDATA #REQUIRED
                 data-format        CDATA #IMPLIED
                 is-replicant  (yes | no) "no"
                 allows-null (yes | no) "no" >
<!ELEMENT keys        (primary-key, foreign-key*) >
<!ELEMENT primary-key        (key-column+) >
<!ELEMENT key-column        (EMPTY)>
<!ATTLIST key-column name CDATA #REQUIRED>
<!ELEMENT foreign-key (column-pair+) >
<!ATTLIST foreign-key parent-group CDATA #REQUIRED >
<!ELEMENT column-pair (EMPTY) >
<!ATTLIST column-pair parent-column        CDATA  #REQUIRED
          child-column CDATA #REQUIRED >
<!ELEMENT create-sql (#PCDATA) >
<!ELEMENT exists-sql (#PCDATA) >
```

The meaning of several elements defined in this DTD follows:

- `<dbschema>` element defines the start of a schema mapping section.
- `<group>` element defines the unit of grouping for a set of fields that will be stored in a single table. The `ddgen.ipl` script (which is run when an administrator uses the `iwsyncdb.ipl -initial` or `iwsyncdb.ipl -ddgen` command) uses the `name` attribute of the group to form part of the base and delta table names in the database.

  When using DAS to deploy data content records, the `name` attribute of the group becomes part of the base and delta table names. The `root-group` attribute indicates whether the group is the root of the group hierarchy. The hierarchy of groups for a particular data content record must have a single root.
- `<attrmap>` element defines the area in which the user specifies what fields in a data content record are mapped to which database columns in the table corresponding to the containing group.
- `<keys>` element allows the user to define primary keys by using the `<primary-key>` element and foreign keys by using the `<foreign-key>` element.
- `<create-sql>` and `<exists-sql>` elements define the SQL code needed to generate the table definition for each group and check for the existence of the table in the database. These elements are optional. If these element are not specified, DataDeploy generates the `CREATE TABLE` statement dynamically from the `<attrmap>` information and uses database vendor-specific SQL statements to check for the existence of a table.

Rules for Implementing User-Defined Database Schemas

This subsection describes:

- General rules for deploying with user-defined database schemas
- Consistency rules for `dbschema.cfg`

General Rules for Deploying with User-Defined Database Schemas

To use user-defined database schemas, the following rules apply:

- You can only deploy data content records and TeamSite extended attributes. In other words, the `<source>` element in a DataDeploy configuration file must contain either `<teamsite-templating-records>` or the `<teamsite-extended-attributes>` element.

- Data category names, data type names, branch names and workarea names cannot contain two consecutive underscores (__).
- If a deployment contains the <dbschema> element, the length of column names specified in the dbschema.cfg file for the data type must be less than or equal to what is allowed by your database vendor. DataDeploy will *not* map long column names to internally generated identifiers that comply with database column name length limitation.
- The <database> element's table-view and clear-table attribute values have no effect when the <dbschema> element is in a DataDeploy configuration file.
- TeamSite Templating must be installed.
- You must use the i-net UNA™ 3.05 JDBC driver to deploy data content records to a Microsoft SQL Server database. Deployment using the JDBC-ODBC bridge may not work properly.

Consistency Rules for dbschema.cfg

dbschema.cfg requires that you implement the following rules:

- Do not use duplicate column names within a <group>/<attrmap> element.
- Do not use duplicate group names within a <dbschema> element.
- If a column is designated as a primary key within a group, a <column> element for that key must exist within that group's <attrmap> element.
- If a column is designated as a foreign key:
  - Its group's <attrmap> element must contain a <column> element whose name matches the name of the child column.
  - Its parent group must exist.
  - Its parent group's <attrmap> element must contain a column whose name matches the name of the parent-group attribute of the child group's <foreign-key>.
- Data-type attributes must match for columns that are specified as foreign keys. In other words, a foreign key must have the same data type (for example, varchar) as its parent key.
- A column that is defined as a primary key cannot contain null values.
- Do not use user-defined database schemas for database sources; only use user-defined database schemas for database destinations. In other words, only use the <dbschema> element inside a <database> element when the <database> element is inside a <destinations> element.

- Do not use narrow tuples. The `options` attribute for the `<teamsite-templating-records>` or `<teamsite-extended-attributes>` element inside the `<source>` element must be set to `wide`.
- If the `update-type` attribute in the `<database>` element is set to `delta`, each `<group>` element inside the `<dbschema>` element must have a `base-table` attribute.
- Specify a primary key for all non-leaf groups in the `dbschema.cfg` file. A group becomes a *leaf* group if its name is not used inside any part of the `<parent-group>` element.
- Do not use the `<select>` and `<update>` elements inside a `<database>` element if the `<database>` element contains a `<dbschema>` element. On the other hand, if a `<database>` element does not contain a `<dbschema>` element, `<database>` must contain `<select>` and `<update>` elements.
- If an `<attrmap>` element inside a `<group>` element has more than one `<column>` definition whose `value-from-field` attribute is set to a replicant field, the value for the specified `value-from-field` must have the same root element. For example, the Treatment and Drug fields in the following example must have the same root element (Treatment List):

```
<group name="drug_list">
    <attrmap>
        <column name="Treatment" data-type="varchar(100)"
            value-from-field="Treatment List/[0-3]/Treatment"
            is-replicant="yes"/>
        <column name="Disease" data-type="varchar(100)"
            value-from-field="Disease " />
        <column name="Drug" data-type="varchar(100)"
            value-from-field="Treatment List/[0-3]/Drug List/[0-4]/Drug"
            is-replicant="yes"/>
    </attrmap>
    ....
</group>
```

- Do not use the syntax that appears similar to regular expressions in the `value-from-field` attribute values (for example, `Treatment List/[0-3]/Drug List/[0-4]/Drug`) unless you are specifying a replicant field. Use this syntax only for replicant fields to indicate the maximum number of replicants for a given node in the XML tree. It cannot be used for any other purpose.

Sample Mappings of dbschema.cfg

A sample mapping of a `dbschema.cfg` file follows:

```
<dbschema>
    <group name = "group1" root-group="yes">
        <attrmap>
            <column name="col1" data-type="varchar(20)"
            value-from-item="item-name1"/>
            <column name="col2" data-type="varchar(20)"
            value-from-item="item-name2"/>
            ........
         </attrmap>

         <keys>
             <primary-key>
                 <key-column name="col1"/>
             </primary-key>
         </keys>
    </group>

    <group name="group2" root-group="no">
        <attrmap>
            <column name="group2col1" data-type="varchar(20)"
                value-from-item="item-name3"
                is-replicant="yes"
                allows-null="no"/>
             .........
         </attrmap>

         <keys>
             <primary-key>
                 <key-column name="group2col1" />
             </primary-key>
             <foreign-key parent-group = "group1" >
                 <column-pair parent-column="col1"
                 child-column =group2col" />
             </foreign-key>
         <keys>
    </group>
</dbschema>
```

A sample mapping of the `dbschema.cfg` file for the medical data capture example template that is distributed with TeamSite Templating would look like this:

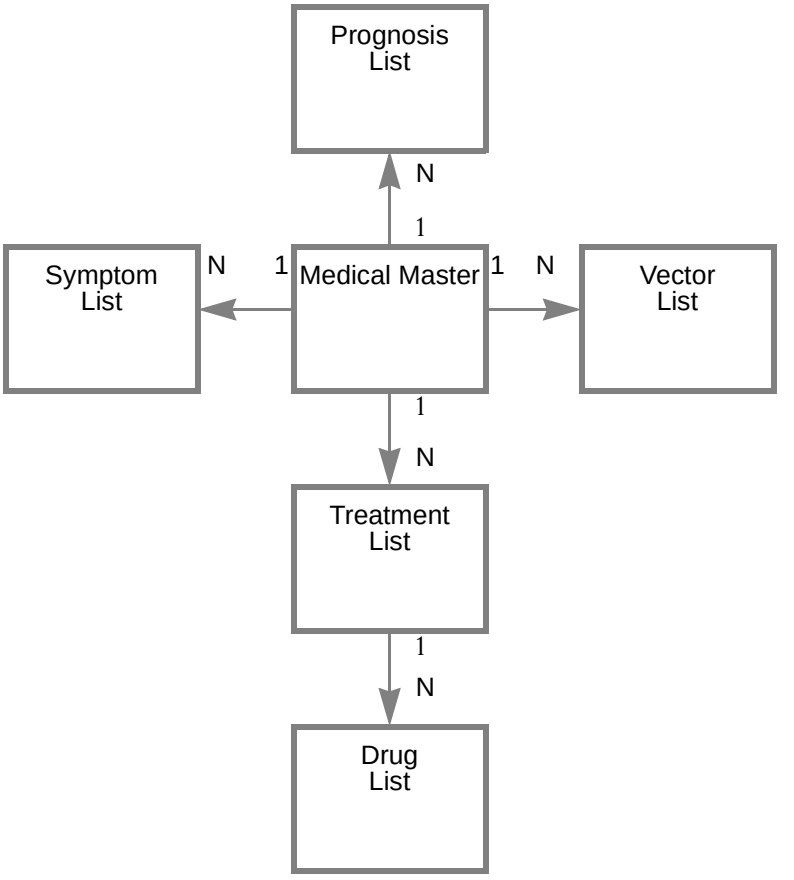
```
<dbschema>
    <group name="medical_master" root-group="yes">
        <attrmap>
            <column name="Disease" data-type="varchar(100)"
            value-from-field="Disease" allows-null="no"/>
            <column name="LatinName" data-type="varchar(100)"
            value-from-field="Latin
            Name" allows-null="no"/>
            <column name="Type" data-type="varchar(15)"
            value-from-field="Type"
            allows-null="no"/>
        </attrmap>

        <keys>
             <primary-key>
                 <key-column name="Disease" />
             </primary-key>
        </keys>
    </group>


    <group name="vector_list" root-group="no">
        <attrmap>
            <column name="Vector" data-type="varchar(40)"
            value-from-field="Vector List/[0-5]/Vector" is-replicant=
            "yes"  />

            <column name="Disease" data-type="varchar(100)"
            value-from-field="Disease"/>
        </attrmap>

        <keys>
            <primary-key>
                <key-column name="Vector"/>
                <key-column name=" Disease"/>
            </primary-key>
```

```
        <foreign-key parent-group=medical_master ">
            <column-pair parent-column="Disease"
            child-column="Disease" />
        </foreign-key>
    </keys>
</group>


<group name="symptom_list" root-group="no">
    <attrmap>
        <column name="Symptom" data-type="varchar(100)"
        value-from-field="Symptom List/[0-5]/Symptom"
        is-replicant="yes"/>
        <column name="Disease" data-type="varchar(100)"
        value-from-field="Disease"/>
    </attrmap>

    <keys>
        <primary-key>
            <key-column name="Symptom"/>
            <key-column name=" Disease"/>
        </primary-key>
        <foreign-key parent-group="medical_master">
            <column-pair parent-column=
            "Disease" child-column="Disease"/>
        </foreign-key>
    </keys>
    </group>


    <group name="prognosis_list" root-group="no">
        <attrmap>
            <column name="Prognosis" data-type="varchar(100)"
            value-from-field="Prognosis List/[0-3]/Prognosis"
            is-replicant="yes"/>
            <column name="Disease" data-type="varchar(100)"
            value-from-field="Disease"/>
            </attrmap>
    <keys>
        <primary-key>
            <key-column name="Prognosis"/>
```

```
            <key-column name=" Disease"/>
        </primary-key>
        <foreign-key parent-group="medical_master">
            <column-pair parent-column="Disease" child-column=
            "Disease" />
        </foreign-key>
    </keys>
</group>

<group name="Treatment_list" root-group="no">
    <attrmap>
        <column name="Treatment" data-type="varchar(100)"
        value-from-field="Treatment List/[0-3]/Treatment"
        is-replicant="yes"/>
        <column name="Disease" data-type="varchar(100)"
        value-from-field="Disease"/>
    </attrmap>

    <keys>
        <primary-key>
            <column-key name="Treatment"/>
            <column-key name="Disease" />
        </primary-key>
        <foreign-key parent-group="medical_master">
            <column-pair parent-column="Disease" child-column=
            "Disease" />
        </foreign-key>
    </keys>
</group>

<group name="drug_list" root-group="no">
    <attrmap>
        <column name="Treatment" data-type="varchar(100)"
        value-from-field="Treatment List/[0-3]/Treatment"
        is-replicant="yes"/>
        <column name="Disease" data-type="varchar(100)"
        value-from-field="Disease"/>
        <column name="Drug" data-type="varchar(100)"
        value-from-field="Treatment List/[0-3]/Drug List/[0-4]/Drug"
        is-replicant="yes"/>
    </attrmap>
```

```
        <keys>
            <primary-key>
                <key-column name="Drug" />
                <key-column name="Disease"/>
                <key-column name="Treatment" />
            </primary-key>
            <foreign-key parent-group="Treatment_list" />
                <column-pair parent-column="Disease"
                child-column="Disease" />
                <column-pair parent-column=" Treatment"
                child-column=" Treatment" />
             </foreign-key>
         </keys>
    </group>
</dbschema>
```

The following figure shows a symbolic representation of the table relationships generated by the previous code:



Symbolic Representation of Table Relationships

Note: *N* is a variable representing the number of rows in a table that contain a given data type. A 1-to-*N* relationship represents a one-to-many relationship.

Sample of basearea Deployment Section

A sample of the `basearea` deployment section in the generated DataDeploy configuration file (*`data_type`*`_dd.cfg`) follows:

```
<deployment name="basearea">
            <source>
                <!-- Pull data tuples from TeamSite Templating DCR's -->
                <teamsite-templating-records
                        options = "wide"
                        area    = "\$mybasearea">
                        <path name = "\$mytemplatepath"
                              visit-directory = "deep" />
                </teamsite-templating-records>
            </source>
            <destinations>
                <!—Oracle 8i -->
                <database db           = "hostname:1524:ORCL"
                          user         = "system"
                         password     = "manager"
                          vendor       = "oracle"
update-type   = "base"
state-field   = "state">

<dbschema>
    <group name="medical_master"
           table= "\$mybasetablenameprefix^__medical_master__\
                  $mybasetablenamesuffix" root-group="yes">
        <attrmap>
            <column name="Path"    data-type="varchar(255)"
            value-from-field="path"/>
            <column name="state"    data-type="varchar(25)"
            value-from-field="state"/>
            <column name="Disease" data-type="varchar(100)"
            value-from-field="Disease"/>
            <column name="LatinName" data-type="varchar(100)"
            value-from-field="Latin Name"/>
            <column name="Type" data-type="varchar(15)"
            value-from-field="Type"/>
        </attrmap>
```

```
        <keys>
            <primary-key>
                <key-column name="Disease" />
            </primary-key>
        </keys>

        <create-sql>
                                  CREATE TABLE
\$mybasetablenameprefix^__medical_master__\$mybasetablenamesuffix
                                  (
                                         path        varchar(255) NOT NULL,
                                         state varchar(255) NOT NULL,
                                         disease varchar(100) NOT NULL,
                                         latinname varchar(100),
                                         type        varchar(15),
                                         CONSTRAINT
\$mybasetablenameprefix^__medical_master__\$mybasetablenamesuffix_key
PRIMARY KEY(disease)
                                  )
        </create-sql>

        <exists-sql>
        SELECT TABLE_NAME FROM USER_TABLES WHERE TABLE_NAME =
'\$mybasetablenameprefix^__medical_master__\$mybasetablenamesuffix'
        </exists-sql>

    </group>


    <group name="vector_list"
           table="\$mybasetablenameprefix^__vector_list__\
                 $mybasetablenamesuffix" root-group="no">
    <attrmap>
        <column name="Vector" data-type="varchar(40)"
        value-from-field="Vector List/[0-5]/Vector" is-replicant="yes"/>
        <column name="Disease" data-type="varchar(100)"
        value-from-field="Disease"/>
        <column name="Path"    data-type="varchar(255)"
        value-from-field="path"/>
    </attrmap>
```

```
    <keys>
        <primary-key>
            <key-column> name="Vector"/>
            <key-column> name="Disease"/>
        </primary-key>

        <foreign-key parent-group="medical_master">
            <column-pair parent-column="Disease"
            child-column="Disease"/>
        </foreign-key>
    </keys>

    <create-sql>

                                CREATE TABLE
\$mybasetablenameprefix^__vector_list__\$mybasetablenamesuffix
                                (
                                        disease varchar(100) NOT NULL,
                                        vector varchar(40) NOT NULL,
                                       path        varchar(255) NOT NULL,
                                        CONSTRAINT
\$mybasetablenameprefix^__vector_list__\$mybasetablenamesuffix_key
 PRIMARY KEY(disease,vector)
                                )
                </create-sql>

                <exists-sql>
                SELECT TABLE_NAME FROM USER_TABLES WHERE TABLE_NAME =
 '\$mybasetablenameprefix^__vector_list__\$mybasetablenamesuffix'
                </exists-sql>

    </group>


    <group name="symptom_list"
           table="\$mybasetablenameprefix^__symptom_list__\
                  $mybasetablenamesuffix" root-group="no">
        <attrmap>
            <column name="Symptom" data-type="varchar(100)"
            value-from-field="Symptom List/[0-5]/Symptom"
            is-replicant="yes"/>
```

```
            <column name="Disease" data-type="varchar(100)"
            value-from-field="Disease"/>
            <column name="Path" data-type="varchar(255)"
            value-from-field="Path"/>
        </attrmap>

        <keys>
            <primary-key>
                <key-column name="Symptom"/>
                <key-column name="Disease"/>
            </primary-key>
            <foreign-key parent-group="medical_master">
                <column-pair parent-column="Disease
                child-column="Disease />
            </foreign-key>
        </keys>

                <create-sql>
                                CREATE TABLE
\$mybasetablenameprefix^__symptom_list__\$mybasetablenamesuffix
                                (
                                        disease varchar(100) NOT NULL,
                                        symptom varchar(40) NOT NULL,
                                        path varchar(255) NOT NULL,
                                        CONSTRAINT
\$mybasetablenameprefix^__symptom_list__\$mybasetablenamesuffix_key
PRIMARY KEY(disease,symptom)
                                )

                </create-sql>

                <exists-sql>
                SELECT TABLE_NAME FROM USER_TABLES WHERE TABLE_NAME =
 '\$mybasetablenameprefix^__symptom_list__\$mybasetablenamesuffix'
                </exists-sql>

    </group>


    <group name="prognosis_list"
           table="\$mybasetablenameprefix^__prognosis_list__\
```

```
                $mybasetablenamesuffix" root-group="no">
        <attrmap>
            <column name="Prognosis" data-type="varchar(100)"
            value-from-field="Prognosis List/[0-3]/Prognosis"
            is-replicant="yes"/>
    <column name="Disease" data-type="varchar(100)"
    value-from-field="Disease"/>
    <column name="Path" data-type="varchar(255)"
    value-from-field="Path"/>
       </attrmap>

       <keys>
           <primary-key>
               <key-column name="Prognosis"/>
               <key-column name="Disease"/>
           </primary-key>
           <foreign-key parent-group="medical_master">
               <column-pair parent-column="Disease
               child-column="Disease"/>
           </foreign-key>
               </keys>

                <create-sql>
                                CREATE TABLE
\$mybasetablenameprefix^__prognosis_list__\$mybasetablenamesuffix
                                (
                                        disease varchar(100) NOT NULL,
                                        prognosis varchar(40) NOT NULL,
                                        path varchar(255) NOT NULL,
                                        CONSTRAINT
\$mybasetablenameprefix^__prognosis_list__\$mybasetablenamesuffix_key
 PRIMARY KEY(disease,prognosis)
                                )
                </create-sql>

                <exists-sql>
                SELECT TABLE_NAME FROM USER_TABLES WHERE TABLE_NAME =
 '\$mybasetablenameprefix^__prognosis_list__\$mybasetablenamesuffix'
                </exists-sql>
    </group>
```

```
    <group name="Treatment_list"
           table="\$mybasetablenameprefix^__treatment_list__\
                 $mybasetablenamesuffix" root-group="no">
        <attrmap>
            <column name="Treatment" data-type="varchar(100)"
            value-from-field="Treatment List/[0-3]/Treatment"
            is-replicant="yes"/>
            <column name="Disease" data-type="varchar(100)"
            value-from-field="Disease"/>
            <column name="Path" data-type="varchar(255)"
            value-from-field="Path"/>
       </attrmap>

       <keys>
           <primary-key>
               <key-column name="Treatment" />
               <key-column> name="Disease"/>
           </primary-key>
           <foreign-key parent-group="medical_master" >
               <column-pair parent-column="Disease"
               child-column="Disease" />
           </foreign-key>
       </keys>
       <create-sql>
                                  CREATE TABLE
\$mybasetablenameprefix^__treatment_list__\$mybasetablenamesuffix
                                  (
                                          disease varchar(100) NOT NULL,
                                          treatment varchar(40) NOT NULL,
                                          path varchar(255) NOT NULL,
                                          CONSTRAINT
\$mybasetablenameprefix^__treatment_list__\$mybasetablenamesuffix_key
 PRIMARY KEY(disease,treatment)
                                  )

        </create-sql>
        <exists-sql>
        SELECT TABLE_NAME FROM USER_TABLES WHERE TABLE_NAME =
'\$mybasetablenameprefix^__treatment_list__\$mybasetablenamesuffix'
        </exists-sql>
```

```
      </group>

    <group name="drug_list"
           table="\$mybasetablenameprefix^__drug_list__\
                 $mybasetablenamesuffix" root-group="no">
        <attrmap>
            <column name="Treatment" data-type="varchar(100)"
            value-from-field="Treatment List/[0-3]/Treatment"
            is-replicant="yes"/>
            <column name="Drug" data-type="varchar(100)"
            value-from-field="Treatment List/[0-3]/Drug List/[0-4]/Drug"
            is-replicant="yes"/>
            <column name="Path" data-type="varchar(255)"
            value-from-field="Path"/>
            <column name="Disease" data-type="varchar(100)"
            value-from-field="Disease"/>
        </attrmap>

        <keys>
            <primary-key>
                <key-column name="Drug" />
                <key-column name="Treatment" />
            </primary-key>
            <foreign-key parent-group="Treatment_list">
                <column-pair
                parent-column="Treatment"
                child-column="Treatment"/>
                <column-pair
                parent-column="Disease"
                child-column="Disease"/>
            </foreign-key>
                </keys>
                <create-sql>
                                CREATE TABLE
\$mybasetablenameprefix^__drug_list__\$mybasetablenamesuffix
                                (
                                        drug varchar(100) NOT NULL,
                                        treatment varchar(40) NOT NULL,
                                        Path varchar(255) NOT NULL,
                                        CONSTRAINT
\$mybasetablenameprefix^__drug_list__\$mybasetablenamesuffix_key PRIMARY
```

```
KEY(drug,treatment)
                                    )
                </create-sql>
                <exists-sql>
                SELECT TABLE_NAME FROM USER_TABLES WHERE TABLE_NAME =
'\$mybasetablenameprefix^__drug_list__\$mybasetablenamesuffix'
                </exists-sql>
    </group>

</dbschema>
```

iwsyncdb.ipl Support for User-Defined Database Schemas

The following subsections describe:

- Creating `dbschema.cfg` files.
- Validating `dbschema.cfg` files.

Note: The command options described in this section do not support creation or validation of a `dbschema.cfg` file for a metadata capture configuration file. See "Deploying Data to User-Defined Database Schemas: Support for Metadata Deployment" on page 80 if you wish to deploy TeamSite extended attributes (metadata).

Creating dbschema.cfg Files

Use the following command to create `dbschema.cfg` files for all data types in a given workarea or, optionally, for a particular data type in a particular data category:

`iwsyncdb.ipl -dbschemagen` *`vpath`* `[`*`dcr-type`*`] [-force]`

Details are as follows:

- This command generates `dbschema.cfg` files for each data type configured in *`iw-home`*`/local/config/templating.cfg` under the specified workarea *`vpath`*.
- Data types that are configured to use custom DTDs are skipped.
- Data types that do not have a `datacapture.cfg` file are skipped.

- The optional *dcr-type* setting causes the creation of a dbschema.cfg file for a single data type (rather than all data types in *vpath*).
- The -force option overwrites any existing dbschema.cfg files.

Validating dbschema.cfg Files

You can validate dbschema.cfg files by using:

- iwsyncdb.ipl -ddgen or -initial
- iwsyncdb.ipl -validate

Note: By default, iwsyncdb.ipl -validate is called as part of both the -ddgen and -initial options in iwsyncdb.ipl. Validation errors are recorded in a file called iwvalidate_dbschema.log in the DataDeploy log directory.

Validation of dbschema.cfg Files Using iwsyncdb.ipl -ddgen or -initial

Both of these options ensure that dbschema.cfg files are properly configured. These options perform this validation by calling iwsyncdb.ipl -validate. If iwsyncdb.ipl -validate detects invalid dbschema.cfg files, the process will be terminated and the errors will be recorded in the iwvalidate_dbschema.log file in the DataDeploy log directory. If a dbschema.cfg file is missing, this log file will not record this as an error.

Validation of dbschema.cfg Files Using iwsyncdb.ipl -validate

You can perform validation of dbschema.cfg files for a:

- *vpath* or a particular data type under *vpath*.
- Or for a file specified with a complete path name.

Validating by Using vpath

The syntax for performing validation of files in a *vpath* is as follows:

```
iwsyncdb.ipl -validate vpath [dcr-type]
```

This command validates the dbschema.cfg files for data types configured in *iw-home*/local/config/templating.cfg under the specified workarea *vpath*. The optional *dcr-type* argument specifies that the iwsyncdb.ipl command will validate a single data type's

dbschema.cfg file (rather than all data types' dbschema.cfg files under *vpath*). If a particular data type does not have a dbschema.cfg file, that type is skipped and no errors are generated.

Validating by Using a Complete Path Name

The syntax for performing validation of a file specified by a complete path name is as follows:

```
iwsyncdb.ipl -validate dbschema_file_name
```

This command validates the specified *dbschema_file_name*. The *dbschema_file_name* argument must specify a complete path to a file that contains the <dbschema> element.

Deploying Custom Data Content Records

You can deploy data content records (DCRs) that are based on custom DTDs into user-defined database schemas.

Notes: Custom DCRs can only be deployed to databases. They cannot be deployed into databases that use wide table format.

The following changes to DataDeploy configuration files enable this feature:

- Support for value-from-element and value-from-attribute attributes has been added to the <column> element.
- Support for a custom attribute has been added to the <teamsite-templating-records> element.

The value-from-element and value-from-attribute Attributes

These attributes have been added to the <column> element in DataDeploy configuration files so that you can specify whether a column maps to an element (node) or an attribute of the node.

To illustrate this, assume that the following data from a custom DCR is to be deployed to a user-defined database schema:

```
<press-release>
 <head>
    <title>title1</title>
    <byline author="Chris" location="location1"/>
 </head>
 <body>
    <heading>heading1</heading>
    <paragraph>para1></paragraph>
 </body>
</press-release>
```

To map an element's value to a column (for example, the value for <title>) the <column> element in the DataDeploy configuration file would look like this:

```
<column   name="title"
          data-type="VARCHAR(100) "
          value-from-element="press-release/0/head/0/title/0"/>
```

To map the value of an element's named attribute to a column (for example, the value for the author attribute), the <column> element in the DataDeploy configuration file would look like this:

```
 <column   name="author"
          data-type="VARCHAR(100) "
          value-from-attribute="press-release/0/head/0/byline/0/author"/>
```

Note the /0/ succeeding each element name. Those are instance indicators that specify which instance of the node's value or attribute is mapped. You must specify the maximum number of instances (replicants) for a given node element. Arbitrarily large values can be specified when the maximum number of replicants allowed is unknown.

Using the same example custom DCR snippet as above, mapping values of replicant elements (in this case, `heading` and `paragraph`) would look like this:

```
<column    name="heading"
           data-type="VARCHAR(100) "
           value-from-element="press-release/0/body/0/heading/[0-5] "
           is-replicant="yes"/>
```

To map the values of replicant element attributes:

```
<column    name="paragraph"
           data-type="VARCHAR(100) "
           value-from-attribute="press-release/0/body/0/paragraph/[0-5] "
           is-replicant="yes"/>
```

The `dbschem.cfg` file created for the `PressRelease` custom DTD example shipped with TeamSite Templating would look like:

```
<dbschema>
 <group name="pr_master" table="pr_master" root-group="yes">
    <attrmap>
       <column name="Path"    data-type="varchar(255)" value-from-
field="path"/>
       <column name="state"    data-type="varchar(25)" value-from-
field="state"/>
       <column name="title" data-type="varchar(100)" value-from-
element="press-release/0/head/0/title/0"/>
       <column name="author" data-type="varchar(100)" value-from-
attribute="press-release/0/head/0/byline/0/author"/>
       <column name="location" data-type="varchar(15)" value-from-
attribute="press-release/0/head/0/byline/0/location"/>
    </attrmap>

    <keys>
       <primary-key>
          <key-column name="title"/>
       </primary-key>
    </keys>

 </group>
```

```
 <group name="pr_body" table="pr_body">
    <attrmap>
       <column name="heading" data-type="varchar(40)" value-from-
element="press-release/0/body/0/heading/[0-5]" is-replicant="yes"/>
       <column name="paragraph" data-type="varchar(100)" value-from-
element="press-release/0/body/0/paragraph/[0-5]" is-replicant="yes"/>
       <column name="title" data-type="varchar(100)" value-from-
element="press-release/0/head/0/title/0"/>
    </attrmap>

    <keys>
       <primary-key>
          <key-column name="title"/>
          <key-column name="heading"/>
          <key-column name="paragraph"/>
       </primary-key>

       <foreign-key parent-group="pr_master" ri-constraint-rule=" ON
DELETE CASCADE ">
          <column-pair parent-column="title" child-column="title"/>
       </foreign-key>
    </keys>
 </group>
</dbschema>
```

The custom Attribute

A `custom` attribute has been added to the `<teamsite-templating-records>` element. To deploy custom DCRs, you must set the value of that attribute to `yes`.

Deploying Data to User-Defined Database Schemas: Support for Metadata Deployment

You can deploy TeamSite metadata into a user-defined database schema in the following modes:

- Standalone
- DAS

Standalone Mode

Ensure that the `update-type` attribute in the `<database>` section of your DataDeploy configuration file has the value of `standalone`. The `update-type` attribute has a default value of `standalone` if you have not explicitly set the value of this attribute.

To deploy metadata with user-defined database schemas in standalone mode:

1. Generate a default `dbschema.cfg` file for a metadata `datacapture.cfg` file that is located at *`iw-home`*`/local/config/datacapture.cfg` by using the following command:

   ```
   iwsyncdb.ipl -mdcdbschemagen [-force]
   ```

   This command generates a `dbschema.cfg` file in the following path: *`iw-home`*`/local/config/dbschema.cfg`.

2. Insert the contents of the generated `dbschema.cfg` file into the `<database>` section of your DataDeploy configuration file.

3. Deploy metadata by using the `iwdd.ipl` command. See "iwdd.ipl Command" on page 159 for full details.

Note: Use `iwsyncdb.ipl -mdcdbschemagen` only for metadata `datacapture.cfg` files based on an Interwoven DTD, `datacapture4.5.dtd` or later.

The `iwsyncdb.ipl -dbschemagen` command does *not* support generating a `dbschema.cfg` file for a medata capture configuration file.

DAS Mode

To deploy metadata with user-defined database schemas in DAS mode:

1. Generate a default `dbschema.cfg` file for a metadata `datacapture.cfg` file that is located at *`iw-home`*`/local/config/datacapture.cfg` by using the following command:

   ```
   iwsyncdb.ipl -mdcdbschemagen [-force]
   ```

2. Create an `mdc_ddcfg_uds.template` file in the *`dd-home`*`/conf` directory.
3. Run `iwsycdb.ipl -mdcddgen [-force]`.

   An *`iw-home`*`/local/config/mdc_dd.cfg` file is created.

Deploying Data from an External Data Source

This section is intended for advanced users or by Interwoven Professional Services personnel. Persons intending to implement the External Data Source feature must install Java Development Kit 1.1 or greater and have good working knowledge of the following:

- Java programming
- Java Development Kit (JDK)
- Java Database Connectivity (JDBC)

DataDeploy supports the inclusion of dynamic (computed at transaction time) values and values from external database objects in deployments. Such values can be used, for instance, to construct a sequence column for each row in the table.

The External Data Source feature supports deployments with user-defined schemas only. The External Data Source feature cannot be used with deployments to databases that use wide table format. External Data Source can be used when the following are deployed in either standalone or DAS mode:

- Metadata
- DCRs
- Custom DCRs

The External Data Source feature is implemented in the form of a programmatic interface. The interface definition is as follows. Additionally, the Java class for the `IWExternalDataSource` interface is installed in the *dd-home*`/conf` directory.

```
package com.interwoven.dd100.dd;

import java.sql.*;
import java.util.*;

public interface IWExternalDataSource
{
    public static final int kOpCodeInsert = 1;
    public static final int kOpCodeUpdate = 2;
    public static final int kOpCodeDelete = 3;
    public static final int kOpCodeQuery  = 4;
public static final double kProtocolVersion10 = 1.0;

public abstract double GetProtocolVersion();
/*
Implementation of this method must return 1.0 or
kProtocolVersion10. If this method returns any other value or
null deployment will be terminated abnormally.
*/


public abstract String GetExternalValue(Connection conn,
String tableName,
        String columnName, int opCode,
        Hashtable tuple, boolean isReplicant);
    /*
    The main interface method that will be invoked by DataDeploy
    to get the value for a column that has been marked to have
    data generated/returned from an external source

Parameters:
    conn - database connection. This is not the same connection
    that DataDeploy uses to perform the deployment.

    tableName - name of the table that has the column to contain
     externally generated value
    columnName - name of the column for which value is to be
     generated by the external source
    opCode - indicates type of operation that will be performed
     by DD on the table
```

```
        1 - insert
        2 - update
        3 - delete
        4 - query (SELECT)

    tuple - hashtable that contains the values for other columns.
     column name is the lookup key.

    isReplicant - true if the column for which value requested is
     mapped to a replicant field
           - false otherwise

        This function should always return the intended value for the
    column as a String. DataDeploy would perform appropriate
    conversion of the String depending on the target column's
    datatype.
        Important: Implementation of this method should take care of
    re-entrancy. This method may be invoked by DataDeploy
    multiple times for the same opcode. For example, when
    DataDeploy  inserts a row into the database, there is a
    preparation stage and there is a second
    stage that performs actual insert. This method should return
    the same value in both the cases. One way of achieving that
    is to look at the "path" attribute value in the tuple
    object, in conjunction with the opCode value.
        When this method is being called when DataDeploy needs to
    select the row, because the original value was supplied by
    this method, it needs to check the database to identify
    the value correctly and return it to DataDeploy.
        Note that any modifications to the key-value pairs in the
    tuple object aren't propagated or used by DataDeploy as it
    supplies only a copy of the tuple object rather than the
    object that it uses to perform the deployment.
    Similarly, the Connection object supplied to this method
    is not the same connection that DataDeploy uses to
    perform the  deployment.
        For DAS deployments, the tableName value supplied to this
    method would be the 'mapped'value if the actual table name
    exceeded the length that the database supports. This method
    can query the IWOV_IDMAPS table to get the original name for
    the supplied mapped name.

    */
}
```

To implement the External Data Source feature, you must write a Java class that implements the above interface (for example, both the abstract methods `GetExternalValue()` and `GetProtocolVersion()`.) See the comments interspersed throughout the definition above for the syntax and semantics of these two methods.

DataDeploy loads the your Java class only once, even if that same class is being used to generate values for multiple columns in various tables. It is the responsibility of the callout implementation to determine, according to the parameters supplied to the `GetExternalValue()` method, which values will be returned.

After you have implemented the Java class as described above, do the following:

1. Compile the Java class.

   During this phase *`dd-home`*`/lib/dd.jar` must be included in the `-classpath` command line option for the Java compiler, or it should be included in the `CLASSPATH` environment variable.

2. Unit test the implementation independent of DataDeploy.

3. Create a `JAR` file for the implementation and copy it to *`dd-home`*`/lib` directory

4. Edit *`dd-home`*`/bin/iwdd.ipl` to include the above `JAR` file in the `datadeploy_jarfiles` variable in `iwdd.ipl`.

5. Add a `<column>` element that defines the column for the dynamic (or external) value to one of the following files:
   - The DataDeploy configuration file that will be used for deployments.
   - The `dbschema.cfg` file if you want to use External Data Source with deployments that occur in DAS mode.

   Using the following example External Data Source interface, such a `<column>` element would look like this:

```
<column name="column1" data-type="VARCHAR(100)" value-from-
callout="iwov:dd:java:com.interwoven.datadeploy.sample.IWDataDeploySample"/>
```

Note that `iwov:dd:java` is a prefix that must precede the class and package name, here represented by `com.interwoven.datadeploy.sample.IWDataDeploySample`.

Example Implementation of the External Data Source Interface

This example can also be found in *dd-home*/conf/IWDataDeploy.java.

```
/*****************************************************************
    This is a sample implementation to demonstrate how DataDeploy
    can interact with a user-defined Java class to supply a value
    for a column during database deployment.

    This Java file should be compiled using JDK1.1 or later.
    CLASSPATH should include $dd-home/lib/dd.jar file. If you are
    using an Integrated Development Environment, refer to manuals
    of that product on how to set the CLASSPATH.

    Once the implementation compiles successfully, you must
    create a Java Archive for this class file and include the
    name of that archive in the CLASSPATH before invoking
    DataDeploy.

    Refer to IWExternalDataSource.java for interface description.

*****************************************************************/

package com.interwoven.datadeploy.sample;

/*
You must have $dd-home/lib/dd.jar in the CLASSPATH or it must be specified in the
-classpath argument to javac.
*/

//import the Interface definition for the Java Callout
import com.interwoven.dd100.dd.IWExternalDataSource;

//import other stuff needed
import java.sql.*;
import java.util.*;


public class IWDataDeploySample implements
com.interwoven.dd100.dd.IWExternalDataSource
{

    // we will use a random generator to generate unique values
```

```
    // for each DCR
    //that gets deployed
    private static Random fRandom = null;

    // we will use a hashtable to store the values with
    // 'tableName' value as the key
    // and TableData object as the value
    private static Hashtable fValues = null;

    //Constructor
    public IWDataDeploySample()
    {
        //nothing to do at this time
    }

    /*
    GetProtocolVersion

    Implementation of the interface method to establish handshake
    with DataDeploy
    */
    public double GetProtocolVersion()
    {
 /*
     currently only supported interface protocol version is 1.0
     */
 return com.interwoven.dd100.dd.IWExternalDataSource.kProtocolVersion10;
    }

    /*
    GetExternalValue

    Implementation of the interface method to supply values
    */

    public String GetExternalValue(Connection conn,
String tableName, String colName, int opCode,
           Hashtable tuple, boolean isReplicant)
    {
        String path = (String) tuple.get("path");

        //check our internal cache for any previously supplied
        //valuefor the given combination of tablename, colname
        //and path value
        String retVal =
```

```
CheckCache4ExistingValue(tableName,colName,path);

        if (retVal == null){ //not in the cache
            //check the table itself. In case of an update we
            //have to supply the existing value
            retVal =
           CheckTable4ExistingValue(conn,tableName,colName,path);
            if (retVal == null || retVal.length() == 0){//no vaue in the target
table
                //generate a new value
                retVal = generateNewValue();
            }
            //add the newly generated value to our cache
            AddToCache(tableName,colName,path,retVal);
        }
        return retVal;
    }

    /*
    generateNewValue

    Generate a new value. We use a Random generator in this
    example. Alternatively the value could be coming from
    another source. For example from a SEQUENCE in the case of
    Oracle database server.
    */
    private String generateNewValue()
    {
        if (fRandom == null){
            fRandom = new Random();
        }
        Integer i = new Integer(fRandom.nextInt());
        return i.toString();
    }

    /*
    CheckCache4ExistingValue

    Check if we have already generated a value for the current
    combination of tablename, column name and path.
    */
    private String CheckCache4ExistingValue(String tableName,
                    String colName, String path)
    {
        String result = null;
```

```
    if (fValues == null){
        fValues = new Hashtable();
    }

    TableData tableData = (TableData) fValues.get(tableName);
    if (tableData == null){
        tableData = new TableData(tableName);
        fValues.put(tableName,tableData);
    }

    result = tableData.getValue(colName,path);
    return result;
}

/*
AddToCache

Add the current combination to cache.
*/
private void AddToCache(String tableName, String colName,
                        String path, String colValue)
{
    if (fValues == null){
        fValues = new Hashtable();
    }

    TableData tableData = (TableData) fValues.get(tableName);
    if (tableData == null){
        tableData = new TableData(tableName);
        fValues.put(tableName,tableData);
    }

    tableData.putValue(colName,colValue,path);
}

/*
CheckTable4ExistingValue

Check the target table for any existing value for the current
combination
*/
private String CheckTable4ExistingValue(Connection conn,
                String tableName,
                String colName, String path)
```

```
    {
        String query = "SELECT " + colName + " FROM " + tableName
                       + " WHERE PATH = ?";
        String result = "";
        try {
            PreparedStatement stmt =
                conn.prepareStatement(query);
            stmt.setString(1,path);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()){
                result = rs.getString(1);
            }
            rs.close();
            stmt.close();
        } catch (SQLException ex){
            System.out.println("Exception:" + ex.getMessage());
            result = "";
        }
        return result;
    }


    //maintains individual column values that we supply, per
    //table and per path
    private class TableData
    {
        private  String fTableName = null;

//'path' level cache
        private  Hashtable fValues = null;

        /*
        Constructor
        */
        public TableData(String tableName)
        {
            fTableName = tableName;
            fValues = new Hashtable();
        }

        /*
        getValue

        Get any existing value for the given combination
        */
```

```
        public String getValue(String colName, String path)
        {
            String result = null;
            Hashtable forPath = (Hashtable) fValues.get(path);
            if (forPath != null){//no cache for this path, allocate a new one
                result =  (String)forPath.get(colName);
            }
            return result;
        }

        /*
        putValue

        Store the combination
        */
        public void putValue(String colName, String value,
                             String path)
        {
            Hashtable forPath = (Hashtable) fValues.get(path);
            if (forPath == null){//no cache for this path, allocate a new one
                forPath = new Hashtable();
                fValues.put(path,forPath);
            }
            forPath.put(colName,value);
        }
    }
}
```

Deploying Data Pointed to from an URL

You can deploy content pointed to from an URL in the following scenarios:

- When DCRs and TeamSite metadata are deployed in standalone or DAS mode with user-defined schemas.
- When custom DCRs are deployed in standalone or DAS mode with user-defined schemas.

The `<column>` element supports the use of these attributes to enable deployment of content that is pointed to by an URL:

- `is-url`—valid values are `yes` and `no`.
- `value-from-field`—valid values are any DCR element or metadata
- `value`—valid values are URLs that contain a `file:` or `http:` prefix.

Additionally, Binary Large Objects (BLOBs) and Character Large Objects (CLOBs) are supported data types.

The following examples and descriptions illustrate how to construct `<column>` elements for deploying content pointed to from an URL:

Example 1

```
<column name="picture" data-type="BLOB" is-url="yes"
value-from-field="TeamSite/Metadata/Picture"/>
```

During deployment, DataDeploy gets the metadata value for the key `TeamSite/Metadata/Picture` for the file being deployed and treats that value as an URL. DataDeploy then reads the contentpointed to by that URL and deploys the content to the `picture` column which is of data type BLOB (Binary Large Object).

Example 2

```
<column name="picture" data-type="BLOB" is-url="yes"
value="file:///C:/TEMP/MYPHOTO.GIF"/>
```

During deployment, DataDeploy reads the content of the file `C:\TEMP\MYPHOTO.GIF` and deploys it to the `picture` column which is of datatype `BLOB`.

Example 3

```
<column name="page" data-type="CLOB" is-url="yes"
value="http://www.interwoven.com/index.html"/>
```

During deployment, DataDeploy reads the content of file `index.html` from the HTTP server `www.interwoven.com` and deploys it to the `page` column which is of data type `CLOB`.

Example 4

```
<column name="picture" data-type="BLOB" is-url="yes"
value-from-field="PressRelease/Picture"/>
```

During deployment, DataDeploy gets the value for the item `PressRelease/Picture` from the DCR being deployed and treats that value as an URL. It then reads the content pointed to by that URL and deploys the content to the `picture` column which is of data type `BLOB`.

Example 2

```
<column name="mytextdata" data-type="VARCHAR(4000)" is-url="yes"
value="file:///C:/TEMP/MYTEXT.TXT"/>
```

During deployment, DataDeploy reads the content of the file `C:\TEMP\MYTEXT.TXT` and deploys that content to the `mytextdata` column whose datatype is `VARCHAR(4000)`. In this case, the content of the file is assumed to be less than or equal to 4000 bytes.

Deploying Replicant Order Numbers

The `<column>` element supports an `replicant-order-number` attribute so that replicant order numbers can be deployed. DataDeploy inserts order values starting from `1`. Order values are automatically adjusted when replicant fields are updated or deleted. Tables can contain multiple replicant order columns. At least one other column definition in the `<group>` element must contain an `is-replicant` attribute that is set to `yes`.

Note: Do not specify the name of the replicant order column as `order` because that is a SQL reserved word.

To create a replicant order column, define a <column> element in a <group> element that is in a <dbschema> element of the DataDeploy configuration file as follows:

Example:

```
<column name="rep_order" data-type="INTEGER" value-from-field="not-used"
allows-null="yes" replicant-order-column="yes"/>
```

1. Specify the value of the replicant-order-column attribute as yes.
2. Specify the value of the value-from-field attribute as not-used, or as an invalid field name.
3. Specify other attributes (data-type, name) as usual.

Enhancing Data Before Deployment

This section is intended for advanced users or by Interwoven Professional Services personnel. Persons intending to implement this feature must install Java Development Kit 1.1 or greater and have good working knowledge of the following:

- Java programming
- Java Development Kit (JDK)
- Java Database Connectivity (JDBC)

DataDeploy enables the dynamic modification data before it is deployed. The data collected by DataDeploy is modified or augmented with tuples from external sources through a Java-based tuple pre-processing callout. The Java interface definition for this callout is as follows (this definition can also be found in *dd-home*/conf/IWExternalTupleProcessor.java):

```
import java.util.Hashtable;

/*
IWExternalTupleProcessor


Java interface definition that user implements to augument/supplement/instrument a
data tuple object, before it gets deployed by DataDeploy.

User must implement PreProcessTuple method.
```

```
*/


public interface IWExternalTupleProcessor
{

/*
PreProcessTuple

The only method in the interface. DataDeploy supplies the data tuple in the form
of a Hashtable object (keys being the field names). User can modify the tuple
(adding, modifying deleting key-value pairs) and then return the modified tuple
object back to DataDeploy.

Arguments:

area: Points to the "area" attribute value if the tuple was produced  by either
<teamsite-templating-records> or <teamsite-extended-attributes>source. null
otherwise.

basearea: Points to the "basearea" attribute value if the tuple was produced by
either <teamsite-templating-records> or <teamsite-extended-attributes> source when
a differential extraction type is specified. null otherwise.
*/
public abstract Hashtable PreProcessTuple(Hashtable tuple,
 String area,
         String basearea);

}

Sample implementation of IWExternalTupleProcessor is as follows. This example can
also be found under $ddhome/conf directory of a DataDeploy installation:

import com.interwoven.dd100.dd.IWExternalTupleProcessor;

import java.sql.*;
import java.util.*;

/*
TupleProcessorExample

This example implementation of IWExternalTupleProcessor demonstrates how a user
written Java class can supplement data that would be deployed DataDeploy.
```

```
For this example, let's assume that user creates a document in the TeamSite
filesystem and sets metadata on the file. Let's assume that the only Extended
Attribute set by the user on the document is called "book_id".

Assuming that related information to "book_id" value is present in another
repository, user wants to deploy that related information using DataDeploy. This
class assumes that the "related" information is present in another database/table
and retrieves such information from those tables and adds it to the
data tuple. When DataDeploy invokes PreProcessTuple method in this class, the
Hashtable tuple object contains "path", "state" and "book_id" values only.

DataDeploy loads a Tuple preprocessor class only once during a deployment.

When you compile this class you should have $ddhome/lib/dd.jar file in the
classpath. After successful compilation, you should build a .jar file for the
.class file and copy it to the $ddhome/lib directory. And then add the .jar file
name to the datadeploy_jarfiles variable in $ddhome/bin/iwdd.ipl file, before
running the deployment.
*/

public class TupleProcessorExample implements IWExternalTupleProcessor
{
    // default constructor. nothing to do here for this
    // example implementation. Typically, user might want to
    // initialize resources that would be required to retrieve
    // related information. For example, establishing database
    // connection etc.
    public TupleProcessorExample()
    {
    }

    // implement the method defined in the
//IWExternalTupleProcessor
// interface
    public Hashtable PreProcessTuple(Hashtable input,
String area, String basearea)
    {

     /*
the input tuple contains path to the file, it's state
   value and other user set extended attributes, in this
example it would be the book_id value
*/

        // get the book_id value
```

```
        String book_id = (String) input.get("book_id");

        /*
once we get the book_id, go to the database and
retrieve the related information. I am not typing the full code here for
retrieving the related information.
*/

        /*
let's just assume that the related information we
     retrieved has author, ISBN, price single-valued
     attributes. we add them to the tuple here
 */
        input.put("author","interwoven");
        input.put("ISBN", "12345");
        input.put("price", "123.33");

        /*
also assume that the related information has repeating-value attributes reviewers
and reviewer-email. let's assume we have
five reviewers and their email addresses to be added to the tuple see the
associated dd config file how these fields are mapped to database table columns
*/

        for (int i = 0; i < 5; i++){
            input.put("Reviews/"+i+"/Name","reviewer"+i);
            input.put("Reviews/"+i+"/EMail","reviewer_email"+i);
        }
        return input;
    }
}
```

Sample DataDeploy configuration file that demonstrates how to specify Java class name for tuple pre-processing:

```
<!—
This example configuration file demonstrates how user written java class to pre-
process the tuple before DataDeploy deploys the data, is specified in the config
file. Pay attention to external-tuple-processor element The user written class in
this example is TupleProcessTest whose source can be found in $ddhome/conf
directory. User can specify multiple java class names separated by comma.
DataDeploy would invoke the PreProcessTuple method in those classes in a serial
fashion, passing the tuple object returned by one class to the next one
-->
```

```
<data-deploy-configuration>
<external-tuple-processor class-name="TupleProcessorExample"/>
    <client>
 <!-- This deployment dumps EA data from a TeamSite area to
several different database destinations               -->
        <deployment name="ea-to-db">
            <source>
                <!-- Pull data tuples from TeamSite EA's -->
                <teamsite-extended-attributes
                        options="full,wide"
                        area="/default/main/br1/WORKAREA/wa1">
                    <path name="test1.txt" />
                </teamsite-extended-attributes>
            </source>

            <destinations>
                <database db="localhost:1521:oracledb"
          user="user"
          password="password"
          vendor="oracle"
          update-type="standalone">
     <dbschema>
        <group name="book_master" root-group="yes" table="book_master">
        <attrmap>
           <column name="path" data-type="varchar(255)" value-from-field="path"
allows-null="no"/>
           <column name="author" data-type="varchar(255)" value-from-
field="author" allows-null="no"/>
           <column name="ISBN" data-type="varchar(255)" value-from-field="ISBN"
allows-null="no"/>
           <column name="book_id" data-type="varchar(255)" value-from-
field="book_id" allows-null="no"/>
        </attrmap>
        <keys>
           <primary-key>
              <key-column name="book_id"/>
           </primary-key>

            </keys>
        </group>

        <group name="book_detail" root-group="no" table="book_detail">
        <attrmap>
            <column name="book_id" data-type="varchar(255)" value-from-
field="book_id" allows-null="no"/>
```

```
            <column name="Reviewer_name" data-type="varchar(255)" value-from-
field="Reviews/[0-4]/Name" allows-null="no" is-replicant="yes"/>
            <column name="Reviewer_Email" data-type="varchar(255)" value-from-
field="Reviews/[0-4]/EMail" allows-null="no" is-replicant="yes"/>
        </attrmap>

        <keys>
           <primary-key>
              <key-column name="book_id"/>
              <key-column name="Reviewer_name"/>
           </primary-key>

           <foreign-key parent-group="book_master">
              <column-pair parent-column="book_id" child-column="book_id"/>
           </foreign-key>
        </keys>
       </group>
    </dbschema>
              </database>
            </destinations>
        </deployment>
    </client>
</data-deploy-configuration>
```

To use the tuple pre-processor callout:

1. Write a Java class that implements the `IWExternalTupleProcessor` Java interface. Implement the `PreProcessTuple()` method as required by the interface definition.
2. Compile the Java class and unit test it.
3. Create a JAR file for the compiled Java class.
4. Copy that JAR file to *`dd-home`*`/lib`.
5. Add the JAR file name to *`dd-home`*`/lib/iwdd.ipl`.
6. Write the DataDeploy configuration file
7. Specify the `<external-tuple-processor>` element and the value of the `class-path` attribute in that configuration file as shown in the above example.

Deploying a Non-replicant Comma Separated List of Values as Replicant Values

Multiple values entered into a field of a data capture form are often stored in the resulting DCR as a comma separated list of values rather than as separate `<item>` elements within a `<replicant>` item. The values in comma separated lists can be deployed as replicant values. That is, those values can be deployed into multiple rows in a table rather than into a single row. This is accomplished by utilizing the `list-field` and `list-to-replicant` attributes of the `<column>` element in a database schema definition. When these two attributes are present, DataDeploy processes the data before the actual deployment takes place and expands the list of values.

For example, assume that a DCR created from the following DCT is to be deployed. Note that a comma separated list of reviewers can be entered for the `Reviewers` item.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE datacapture SYSTEM "datacapture5.0.dtd">

<data-capture-requirements type="content" name="book">
    <!-- data-capture-requirements elements contain area elements -->
    <ruleset name="Book Information">

       <description>
            This allows for the entry of details relating to a book.
        </description>

       <item name="Title">
            <database data-type="VARCHAR(100)" />
            <text required="t" maxlength="100" />
        </item>

       <item name="Author">
            <database data-type="VARCHAR(40)" />
            <text required="t" maxlength="40" />
        </item>

       <item name="ISBN">
            <database data-type="VARCHAR(20)" />
            <text required="t" maxlength="20" />
       </item>
```

```
        <item name="Price">
             <description>dollars and cents</description>
             <database data-type="REAL" />
             <text required="t" maxlength="7" validation-regex="^[0-
9]+\.[0-9]{2}$" />
          <!-- validation-regex="^[0-9]+\.[0-9]{2}$" means there is a
match if the entire string contains 1 or more digits
                followed by a . followed by 2 digits -->
         </item>


         <item name="Reviewers">
             <description>Reviewer names separated by comma</description>
             <database data-type="VARCHAR(255)" />
             <text required="t" maxlength="255"  />
         </item>

    </ruleset>
</data-capture-requirements>
```

The following database schema definition is generated for this DCT when you run `iwsyncdb.ipl -dbschemagen`:

```
<dbschema>
   <group name="book" root-group="yes">
      <attrmap>
         <column name="Title" data-type="VARCHAR(100)" value-from-
field="Title" allows-null="no"/>
         <column name="Author" data-type="VARCHAR(40)" value-from-
field="Author" allows-null="no"/>
         <column name="ISBN" data-type="VARCHAR(20)" value-from-
field="ISBN" allows-null="no"/>
         <column name="Price" data-type="REAL" value-from-field="Price"
allows-null="no"/>
         <column name="Reviewers" data-type="VARCHAR(255)" value-from-
field="Reviewers" allows-null="no"/>
      </attrmap>
      <keys>
         <primary-key>
            <key-column name="Title"/>
```

```
         </primary-key>
      </keys>
   </group>
</dbschema>
```

DCRs created and deployed by using the above DCT and associated database schema would create a single table in the database as follows:

Title	Author	ISBN	Price	Reviewers
Interwoven	Author1	12345	100.00	R1,R2,R3,R4

To deploy the same data such that the comma separated list of reviewers is stored in a separate table as replicant values, modify the database schema as follows:

```
<dbschema>
   <group name="book" root-group="yes">
      <attrmap>
         <column name="Title" data-type="VARCHAR(100)" value-from-
field="Title" allows-null="no"/>
         <column name="Author" data-type="VARCHAR(40)" value-from-
field="Author" allows-null="no"/>
         <column name="ISBN" data-type="VARCHAR(20)" value-from-
field="ISBN" allows-null="no"/>
         <column name="Price" data-type="REAL" value-from-field="Price"
allows-null="no"/>
 </attrmap>
      <keys>
         <primary-key>
            <key-column name="Title"/>
         </primary-key>
      </keys>
   </group>

      <group name="reviewers" root-group="no">
      <attrmap>
         <column name="Title" data-type="VARCHAR(100)" value-from-
field="Title" allows-null="no"/>
```

```
    <column name="Reviewers" data-type="VARCHAR(255)" list-to-
replicant="yes" list-field="Reviewers" value-from-field="Reviewers/[0-4]/
Reviewer" is-replicant="yes" allows-null="yes"/>
</attrmap>
      <keys>
         <primary-key>
            <key-column name="Title"/>
 <key-column name="Reviewers"/>
         </primary-key>
    <foreign-key parent-group="book">
            <column-pair parent-column="Title" child-column="Title"/>
 </foreign-key>
      </keys>
   </group>

</dbschema>
```

In the above modified database schema, an additional `<group>` element represents the additional table that contains the list of reviewers. Note that the `Reviewers` column definition in that group contains `list-field` and `list-to-replicant` attributes. The `list-field` attribute contains the comma separated list of values, and the `list-to-replicant` indicates to DataDeploy that it must transform that list into multiple values before deployment. A deployment based on the above modified database schema would look like this:

Title	Author	ISBN	Price
Interwoven	Author1	12345	100.00

Title	Reviewers
Interwoven	R1
Interwoven	R2
Interwoven	R3
Interwoven	R4

Other Data Organization Issues

Data Types and Sizes

The default data type for deployed data is `VARCHAR(255)`. You can set different data types, or a different size for `VARCHAR`, in the DataDeploy configuration file. See Item 11, “Rows to update” in “Sample File Notes” starting on page 115 for more information.

Database Object Name Lengths

To overcome the maximum database object name length imposed by database servers, DataDeploy builds a mapping table called `IWOV_IDMAPS` in the destination database. For each object name that exceeds the maximum length limit for the database, this mapping table establishes a relationship between the original object name and a generated name conforming to the database’s object name length limits. The generated name is then used in place of the original object name in all database transactions. This implementation allows table names, column names, constraint names, and view names to contain any number of characters.

The `IWOV_IDMAPS` table contains three columns: `Type`, `Longid`, and `Shortid`. The `Type` column defines types as follows:

`1`: Table name
`2`: Column name
`3`: View name
`4`: Constraint name

The `Longid` column contains the entire character string for the object as it appears in the original source file. The `Shortid` column contains the generated name conforming to the database’s object length limits. For example, a typical table might appear as follows:

Type	Longid	Shortid
2	INFORMATIONO_PRESENTATIONTITLE	IWC_AA6A93A7161
1	INTRANET_DEPTINFO__DATADPLYBRNCH_STAGING	IWT_106342E4D4C4
3	INTRANET_DEPTINFO__DATADPLYBRNCH_STAGING_VIEW	IWV_AEGF12D4E
4	INTRANET_DEPTINFO__DATADPLYBRNCH_STAGING_CONSTRAINT	IWO_F023AF1290

Because different databases support different maximum object name lengths, the threshold for when a `Shortid` name is generated depends on the database vendor and/or type. DataDeploy uses the values set for the `max-id-value` attribute to determine this threshold. See Item 10, “Database Section” in “Sample File Notes” starting on page 115 for more information. See also “Table Update Details” on page 182.

When deploying to a DB2 database, DataDeploy maps table, column, and view names only when a name exceeds 128 characters, and maps constraint names only when they exceed 18 characters.

If you construct an SQL statement that performs an activity on a table that was created by DataDeploy, and if that table contains any database objects whose names exceed the maximum length, the SQL statement must first reference the mapping table to determine the actual (`Longid`) object name(s). This requirement applies to all SQL statements, including those not executed through DataDeploy.

Chapter 5

Configuration File Details and Examples

This chapter contains the following detailed information about configuration file contents:

- Which elements are required in each type of configuration file.
- Rules for parameter substitutions within configuration files.
- An annotated sample TeamSite-to-database configuration file.
- A sample TeamSite-to-XML configuration file.
- A sample database-to-database configuration file.
- A sample database-to-XML configuration file.
- A sample XML-to-database configuration file.
- A sample XML-to-XML configuration file.
- The configuration files for "Starting State," "Event 1," and "Event 2" shown on page 38.

Note: If you are using DataDeploy in a non-US English environment, see Appendix D, "Internationalization."

Required Elements

The type of deployment (for example, TeamSite-to-database, TeamSite-to-XML, and so on) determines which configuration file sections are required and which elements can reside in each section. Only a few parameters are actually required within these sections. The rest are optional, making it possible to have short, simple configuration files. Section hierarchy and requirements for each supported type of deployment are as follows. Sections in bold text are required; those in normal text are optional. Indentation shows nesting levels.

TeamSite-to-Database

```
filter
   keep
   discard
substitutions
data-deploy-elements
client
deployment
   substitutions
   exec-deployment
   source
      teamsite-extended-attributes
      teamsite-templating-records
   destinations
      substitutions
      filter
      database
         select
         update
         sql
server
```

TeamSite-to-XML

```
filter
   keep
   discard
substitutions
client
data-deploy-elements
deployment
   substitutions
   exec-deployment
   source
      teamsite-extended-attributes
      teamsite-templating-records
   destinations
      substitutions
      filter
      xml-formatted-data
server
```

Database-to-Database

```
filter
   keep
   discard
substitutions
data-deploy-elements
client
deployment
   substitutions
   exec-deployment
   source
      database
         fields
   destinations
      substitutions
      filter
      database
         select
         update
         sql
server
```

Database-to-XML

```
filter
   keep
   discard
substitutions
data-deploy-elements
client
deployment
   substitutions
   exec-deployment
   source
      database
         fields
   destinations
      substitutions
      filter
      xml-formatted-data
server
```

XML-to-Database

```
filter
   keep
   discard
substitutions
client
data-deploy-elements
deployment
   substitutions
   exec-deployment
   source
      xml-formatted-data
         fields
   destinations
      substitutions
      filter
      database
         select
         update
         sql
server
```

XML-to-XML

```
filter
   keep
   discard
substitutions
data-deploy-elements
client
deployment
   substitutions
   exec-deployment
   source
      xml-formatted-data
         fields
   destinations
      substitutions
      filter
      xml-formatted-data
server
```

Parameter Substitutions

Any parameter string in a configuration file can be named using a parameter substitution. You set parameter string substitutions on the same command line you use to invoke DataDeploy with the `iwdd` command. Syntax is as follows:
"*varname=varvalue*"

After a string is defined on the command line, all occurrences of $*varname* in the configuration file named on the command line are substituted with the string *varvalue*. Do not use the following terms for *varname*; they are keywords for the `iwdd` command and would be interpreted as such:

- `cfg`
- `deployment`
- `iwdd-op`
- `remote-host`
- `remote-port`

Examples of parameter substitution within a configuration file are as follows:

*prefix_string_*$*varname*
$*varname*^*_suffix_string* (where ^ is a concatenator)
*prefix_*$*varname*^*_suffix*

Sample TeamSite-to-Database Configuration File

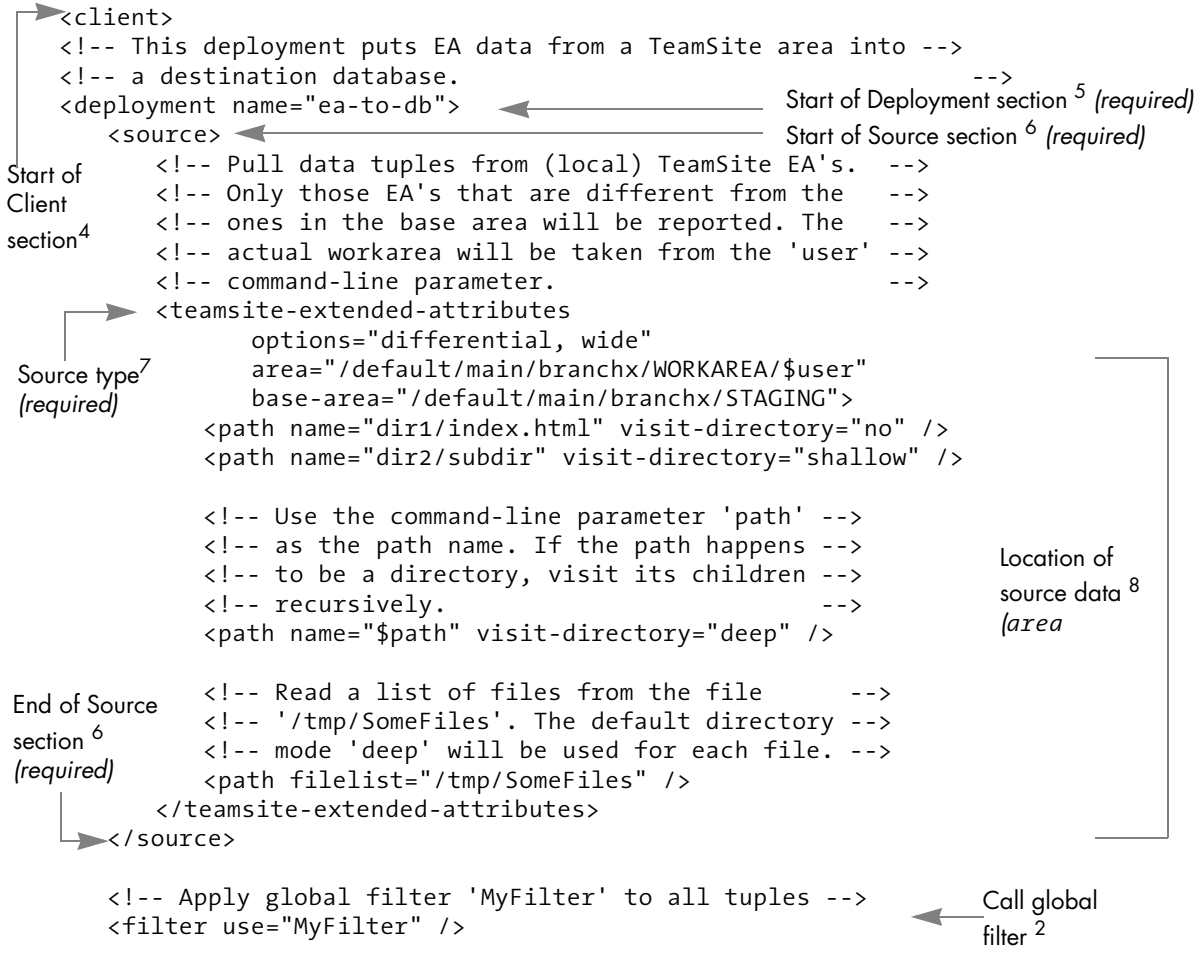
The following sample configuration file shows how to set parameters for a typical TeamSite-to-database deployment. It identifies which parameters are required, shows both global and in-flow usage, and is keyed to a comment table following the file that explains more details about each section and parameter. Most of the elements in this file are also used to define types of deployment other than TeamSite-to-database. For examples of configuration files for these other deployment types, see the sample file sections starting on page 141.

```
<!--Sample DataDeploy configuration file -->
<data-deploy-configuration>
 <data-deploy-elements filepath="/local/iw-home/db.xml"/>
 <filter name="MyFilter">
    <!-- This is a filter that can be used by any deployment -->
    <keep>
       <!-- Any of the following (logical OR):               -->
       <!-- dir2/subdir/index.html, any *.html file in dir1, -->
       <!-- OR anything with key 'guard' AND value 'IGNORE   -->
      <or>
       <field name="path" match="dir2/subdir/index.html" />
       <field name="path" match="dir1/*.html" />
       <and>
          <!-- Must match all of these (logical AND) -->
          <field name="key"   match="guard" />
          <field name="value" match="IGNORE" />
       </and>
      </or>
    </keep>

    <discard>
       <!-- Exclude the file dir1/ignoreme.html, anything     -->
       <!-- with key 'unneededKey', and anything with state  -->
       <!-- DELETED                                          -->
      <or>
       <field name="path"  match="dir1/ignoreme.html" />
       <field name="key"   match="unneededKey" />
       <field name="state" match="DELETED" />
      </or>
    </discard>
 </filter>

 <substitution name="GlobalSubstitution">
    <!-- This substitution can be used by any deployment.                -->
    <!-- It replaces the first occurrence of the string 'foo'            -->
    <!-- in the 'path' field with 'bar', and completely                  -->
    <!-- replaces the 'value' field with the string 'SpecialValue'.-->
    <field name="path" match="foo" replace="bar" />
    <field name="value" replace="SpecialValue" />
 </substitution>
```

Include file [1]

Filter section (global) [2]

Substitution section (global) [3]


<client>
<!-- This deployment puts EA data from a TeamSite area into -->
<!-- a destination database. -->
<deployment name="ea-to-db">
<source>
Start of Deployment section 5 (required)
Start of Source section 6 (required)
Start of Client section4
<!-- Pull data tuples from (local) TeamSite EA's. -->
<!-- Only those EA's that are different from the -->
<!-- ones in the base area will be reported. The -->
<!-- actual workarea will be taken from the 'user' -->
<!-- command-line parameter. -->
<teamsite-extended-attributes
options="differential, wide"
area="/default/main/branchx/WORKAREA/$user"
base-area="/default/main/branchx/STAGING">
Source type7 (required)
<path name="dir1/index.html" visit-directory="no" />
<path name="dir2/subdir" visit-directory="shallow" />
<!-- Use the command-line parameter 'path' -->
<!-- as the path name. If the path happens -->
<!-- to be a directory, visit its children -->
<!-- recursively. -->
<path name="$path" visit-directory="deep" />
Location of source data 8 (area
<!-- Read a list of files from the file -->
<!-- '/tmp/SomeFiles'. The default directory -->
<!-- mode 'deep' will be used for each file. -->
<path filelist="/tmp/SomeFiles" />
End of Source section 6 (required)
</teamsite-extended-attributes>
</source>
<!-- Apply global filter 'MyFilter' to all tuples -->
<filter use="MyFilter" />
Call global filter 2

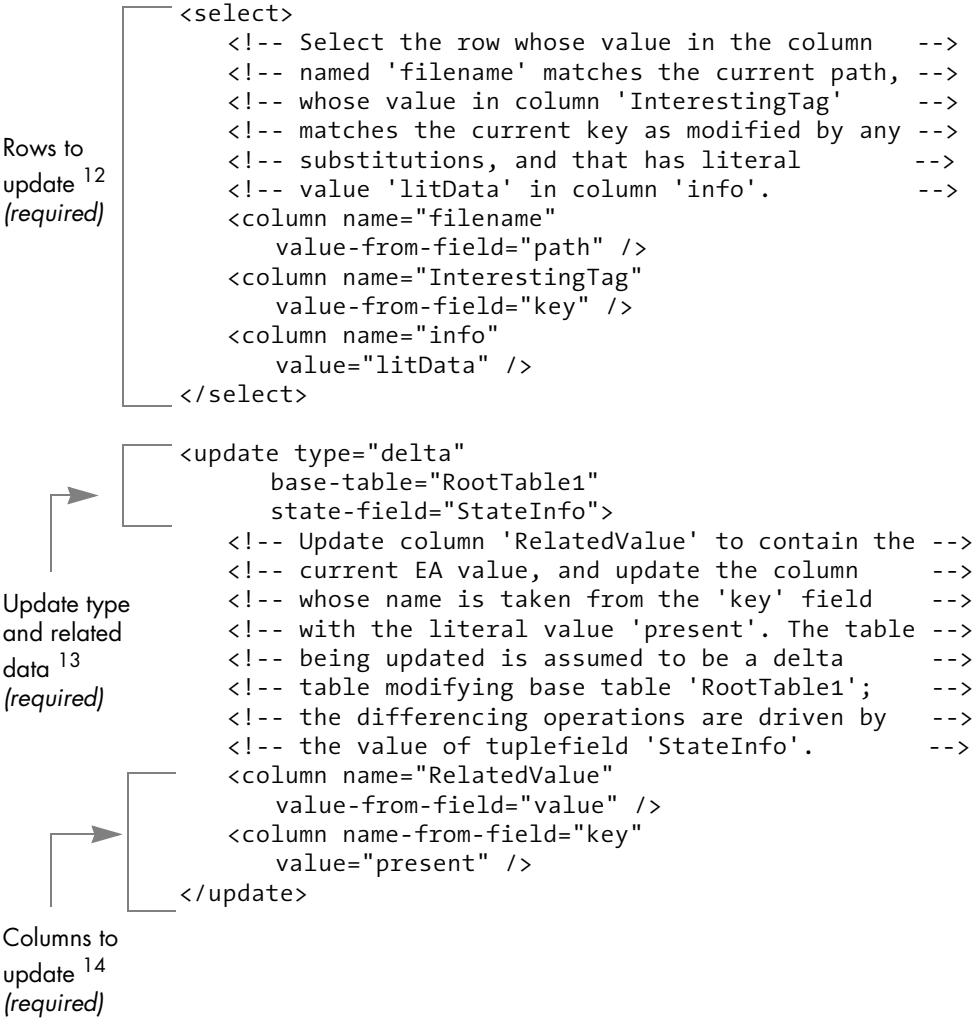
```
<substitution>
   <!-- Modify each tuple according to the following  -->
   <!-- match/replace pairs. In this case: any path   -->
   <!-- that contains the string 'WORKAREA/.../' will -->
   <!-- have the string replaced by 'STAGING/'; any   -->
   <!-- path that contains 'EDITION/abcd' will be     -->
   <!-- replaced with '/This/Special/Path', and any   -->
   <!-- tuple whose key starts with 'BEFORE' will be  -->
   <!-- changed to begin with 'AFTER'.                -->
   <field name="path"
      match="(.*)/WORKAREA/[^/]+/(.*)"
      replace="$1/STAGING/$2" />
   <field name="path"
      match="EDITION/abcd"
      replace="/This/Special/Path" />
   <field name="key"
      match="^BEFORE(.+)"
      replace="AFTER$1" />
</substitution>
<!-- Also apply the substitution 'GlobalSubstitution'       -->
<substitution use="GlobalSubstitution" />

<!-- Start the destinations section.                        -->
<destinations
   host="DDServer.interwoven.com"
   port="1357">
   <!-- Filtered and substituted data will be sent to a     -->
   <!-- DataDeploy server on port 1357 of the machine       -->
   <!-- DDServer.interwoven.com. Then                       -->
   <!-- send some tuples to 'table1' on the database that   -->
   <!-- is located using 'jdbc:remote.machine.com' and      -->
   <!-- provide user 'dba' with password 'ThisIsASecret'.   -->
   <!-- Perform any other activities that are associated    -->
   <!-- with the option 'ea-update'. Timeout is 45 seconds. -->
   <database name="myproductiondb"
      db="host1:1357:db1"
      table="table1"
      vendor="oracle"
      user="dba"
      password="ThisIsASecret"
      timeout="45">
```

Substitution section (in-flow) [9]

Call global substitution [3]

Start of Destinations section [10] *(required)*

Start of Database section and location of destination database [11] *(required)*




```
<select>
   <!-- Select the row whose value in the column   -->
   <!-- named 'filename' matches the current path, -->
   <!-- whose value in column 'InterestingTag'     -->
   <!-- matches the current key as modified by any -->
   <!-- substitutions, and that has literal        -->
   <!-- value 'litData' in column 'info'.          -->
   <column name="filename"
      value-from-field="path" />
   <column name="InterestingTag"
      value-from-field="key" />
   <column name="info"
      value="litData" />
</select>

<update type="delta"
      base-table="RootTable1"
      state-field="StateInfo">
   <!-- Update column 'RelatedValue' to contain the -->
   <!-- current EA value, and update the column     -->
   <!-- whose name is taken from the 'key' field    -->
   <!-- with the literal value 'present'. The table -->
   <!-- being updated is assumed to be a delta      -->
   <!-- table modifying base table 'RootTable1';    -->
   <!-- the differencing operations are driven by   -->
   <!-- the value of tuplefield 'StateInfo'.        -->
   <column name="RelatedValue"
      value-from-field="value" />
   <column name-from-field="key"
      value="present" />
</update>
```

SQL section [15]

```
            <!-- If it is necessary to create a new table for -->
            <!-- this deployment, the following SQL statement -->
            <!-- will be used for that purpose (as opposed to -->
            <!-- a capriciously chosen internal default)      -->
            <sql action="create">
               <!-- This comment should be ignored. However  -->
               <!-- the parameter token in the next line is  -->
               <!-- subject to parameter substitution.       -->
                  CREATE TABLE table1 (
                     Path VARCHAR(300) NOT NULL,
                     KeyName VARCHAR(300) NOT NULL,
                     Value VARCHAR(4000) ,
                     State VARCHAR(4000) ,
                     CONSTRAINT KVP PRIMARY KEY (Path,KeyName)
                  )
            </sql>
         </database>
      </destinations>
   </deployment>

   </client>

   <server>
      <!-- The DataDeploy server will listen on port 1949 of IP -->
      <!-- 204.247.118.99                                       -->
      <bind ip="204.247.118.99" port="1949" />

      <!-- Only accept connections from these hosts -->
      <allowed-hosts>
         <host addr="ddclient.interwoven.com" />
         <host addr="204.247.118.33" />
      </allowed-hosts>

      <!-- Server-specific deployment information -->
      <for-deployment name="ea-to-db">
         <database db="host1:1357:db1"
            user="scott"
            password="tiger" />
      </for-deployment>
   </server>
</data-deploy-configuration>
```

Server section [16]

Sample File Notes

1. **Include File:** You can use `<data-deploy-elements>` to name a file containing data to include by reference. The file named in `<data-deploy-elements>` can contain any number of `<database>`, `<filter>`, and `<substitution>` elements. It must use the same syntax for these elements that the main DataDeploy configuration file uses. See Items 2, 3, and 11 in this section for details. If mutually exclusive attributes are set in the include file and the main DataDeploy configuration file, all are used in the deployment. If conflicting attributes are set in the two files, those set in the main DataDeploy configuration file take precedence.

2. **Filter section (global):** Filters let you explicitly state which tuples will be deployed. The `keep` section contains criteria for selecting which tuples will be deployed, and the `discard` section contains criteria for those which will not be deployed. Both sections use `field` tags. All `field` tags must contain at least one `name/match` attribute pair. When you deploy from TeamSite, `name` must be either `key`, `value`, `path`, or `state`. When you deploy from a source other than TeamSite, `name` can be any be any field name that is valid in the source area. The `match` attribute names a targeted value for `name`. A filter defined in the nesting level shown here and located before the Deployment section will be global. Global filters do not become active until they are called by the `<filter>` element's `use` attribute between the Source and Destinations sections using the syntax shown later in the sample file. Note that filters can also be defined in an include file and then be called by the `use` attribute. If a configuration file does not contain a filter section, all tuples are deployed (limited only by the type of update being performed). A configuration file can contain any number of global filter sections. A configuration file can also contain in-flow filters within a `destinations` section. See Item 10 for details.

3. **Substitution section (global):** Substitutions let you configure DataDeploy to automatically replace character strings or entire fields in a table. Substitutions use `field` tags that must contain at least one `name/replace` attribute pair. As with filters, `name` is either `key`, `value`, `path`, or `state`. The `replace` attribute is the new string that will overwrite the existing string or field.

Two additional attributes, `match` and `global`, are optional. Common usage examples are as follows:

To do this:	Include this line in the Substitution section:
Replace all Value field values with the string `Newvalue`	`<field name="value" replace="Newvalue"/>`
In the Path field, replace first occurrence of `blue` with `red`	`<field name="path" match="blue" replace="red"/>`
In the Path field, replace all occurrences of `blue` with `red`	`<field name="path" match="blue" replace="red" global="yes"/>`
In the State field, replace the first occurrence of `Original` with `NotPresent`	`<field name="key" match="Original" replace="NotPresent"/>`

A substitution defined in the nesting level shown here and located before the Deployment section will be global. Global substitutions do not become active until they are called by the `<substitution>` element's `use` attribute between the Source and Destinations sections using the syntax shown later in the sample file. Note that substitutions can also be defined in an include file and then be called by the `use` attribute. A configuration file can contain any number of global substitution sections.

4. **Client section:** The `client` section lets you specify a set of client-specific parameters and activities. A configuration file that is expected to run on a two-tier system or as a client on a three-tier system must have exactly one client section.

5. **Deployment section:** The `deployment` section is where you assign a name to each deployment, and specify deployment source, destination, and update type. You can have any number of `deployment` sections in a configuration file, and each must have a unique name. The name shown here, `ea-to-db`, is the name you would specify on the command line when you invoke DataDeploy. The `deployment` section is required in all configuration files. The `<exec-deployment>` subelement lets you execute one or more deployments that are defined elsewhere in the same configuration file. Syntax is as follows:

```
<exec-deployment use="dbname" />
```

where *dbname* refers to the name of a database as defined in the name attribute in a <database> element.

6. **Source section:** The source section resides one nesting level inside the deployment section. It is where you name the type of data to extract from TeamSite and the location(s) of that data. Each deployment section must have exactly one source section.

7. **Source type:** The first nesting level within the <source> element contains a subelement defining what type of data is to be extracted from TeamSite. This subelement has the following possible elements:

Subelement	Description
teamsite-templating-records	Used when deploying a TeamSite Templating data content record from TeamSite. Supported options: wide (default), full (default), differential.
teamsite-extended-attributes	Used when deploying anything other than a data content record from TeamSite. Supported options: narrow (default), wide, full (default), differential.
xml-formatted-data	Used when deploying from an XML file. Supported options: narrow (default), wide, full (default), differential.
database	Used when deploying from a database. Supported options: narrow (default), wide, full (default), differential.

Each of the preceding subelements supports three attributes: `options`, `area`, and `base-area`. Details about the `options` attribute are as follows:

`options` Value	Description
`wide`	Creates a wide table based on wide tuples containing any number of key-value pairs. Specified in addition to `differential` or `full`. The `wide` and `narrow` values are mutually exclusive; you cannot specify both within the same element. The `wide` value is the default for the `teamsite-templating-records` element.
`narrow`	Creates a 4-column (narrow) table based on narrow tuples. Specified in addition to `differential` or `full`. The `wide` and `narrow` values are mutually exclusive; you cannot specify both within the same element. The `narrow` value is the default for the `teamsite-extended-attributes`, `xml-formatted-data`, and `database` elements. The `teamsite-templating-records` element does not support the `narrow` value.
`differential`	Instructs DataDeploy to extract just the delta data from a workarea/staging-area comparison. Normally, you specify `differential` when performing a delta update. The `differential` and `full` values are mutually exclusive; you cannot specify both within the same element. The default is `full`.
`full`	Instructs DataDeploy to create a table populated with all of the data from a named area. Normally, you specify `full` when performing a base or standalone update (update types are defined later in the `destinations` section). The `differential` and `full` values are mutually exclusive; you cannot name both as options within the same element. The default is `full`.

To configure an incremental deployment, set the `<teamsite-extended-attributes>` or `<teamsite-templating-records>` elements as follows. The result is a delta table containing the differences between *`vpath1`* and *`vpath2`*.

```
<teamsite-extended-attributes
  options="differential"
  area="vpath1"
  base-area="vpath2"

  ...additional subelements if necessary...

</teamsite-extended attributes>
```

8.Location of source data: The `area` attribute defines the TeamSite workarea, staging area, or edition from which DataDeploy will extract data. This attribute is required in all `deployment` sections. The value of `area` should be the vpath name of the area containing the changes you intend

to deploy. If `differential` is set, you must also supply a vpath value for `base-area`. This value should be the vpath name of the edition or staging area that is the basis for comparison with the workarea you named in `area`. The optional `path` element can have one (but not both) of the following values: `name` or `filelist`. Setting the `name` attribute lets you specify a relative path name to a file or directory in the area(s) you named earlier in `area` (and `base-area` if applicable), or stipulate that the path name will be entered on the command line when you invoke DataDeploy. See "Parameter Substitutions" on page 109 for information about entering path names on the command line. Setting the `filelist` attribute lets you specify a file containing a list of files, and is typically used when you perform a delta update of a workarea containing only a few changed files. If you do not name a `path` value, it defaults to "`.`" and DataDeploy performs a deep search of the directory named in `area` (and `base-area` if applicable). The `visit-directory` attribute lets you specify DataDeploy's level of searching within a directory. The three possible values are `no`, `shallow`, and `deep`. Details are as follows:

Value	Description
`no`	If `path name` is a directory, it is not searched.
`shallow`	If `path name` is a directory, it is searched to the first level.
`deep`	All directories and all subdirectories found in `path name` are searched recursively.

The default value of `visit-directory` is `deep`.

9. **Substitution section (in-flow):** In-flow substitutions let you define substitution rules that apply only to specific parts of a deployment. DataDeploy supports in-flow substitutions within the `deployment` and `destinations` elements. For example, the in-flow substitution shown in the sample configuration file is nested one level inside of the `deployment` element, and therefore applies only to the `ea-to-db` deployment. You can also nest in-flow substitutions one level inside `destinations` elements, in which case the substitution applies only to a specific destination. In-flow substitutions have the same syntax as global substitutions. In addition, in-flow substitutions support a `global` attribute that lets you that lets you control whether the substitution applies to all occurrences or just the first occurrence of the matching pattern.

If `global` is set to `no`, the substitution applies only to the first occurrence. If it is set to `yes`, the substitution applies to all occurrences. For example:

```
<destinations>
   <database . . .>
   <substitution name="SubForThisTarget">
      <field  name="BField" match="from_a"
              replace="to_b"
              global="yes" />
   </substitution>
```

The example shown in the sample configuration file earlier in this chapter uses Perl 5 regular expression syntax for `match` values. A configuration file can contain any number of in-flow substitution sections.

10. **Destinations section:** The `destinations` section resides one nesting level inside the `deployment` section. It is where you name the destination system(s), timeout value, database, and table, and is also where you define the update type. Each `deployment` section can have any number of `destinations` sections, allowing you to designate multiple destinations in a single configuration file. Destination system and timeout details are as follows. Database, table, and update type are explained later in Item 11.

Attribute	Description	Required?	Value Syntax
`host`	Machine name of the DataDeploy server (3-tier systems only).	No	"*host.domain*.com"
`port`	Port on `host` to which data will be sent.	No	"*portnumber*"
`timeout`	How long the client system will wait for a response from the remote host during communication exchange. This tag can also reside in the Database section, in which case it has a different definition. See Item 11 for details.	No	"*seconds*"

You can also nest in-flow filters within a `destinations` element, in which case the filter applies only to that specific destination. For example:

```
<destinations>
   <database . . .>
   <filter name="FilterForThisTarget">
      <discard>
         <field name="AField" match="^DoNotWant/.*"/>
      </discard>
   </filter>
```

In-flow filters have the same syntax as global filters.

11. **Database section:** The first subelement in the `destinations` section defines the type of destination for the data. This subelement can be either `<database>` or `<xml-formatted-data>`, depending on whether the destination is a database or an XML file. See "Sample TeamSite-to-XML Configuration File" on page 141 for an example of `xml-formatted-data` usage. When deployment is to a database, the `<database>` tag and its `name` and `db` attributes are required in all `deployment` sections. A `destinations` section can have any number of `<database>` subelements or a combination of `<database>` and `<xml-formatted-data>` subelements. Syntax for the values `db` and other attributes of the `<database>` tag are as follows:

Attribute	Value	Description	Required?
`name`	Any user-defined database name surrounded by double quotes, for example, `"myproductiondb"`.	Used to reference the database through the `use` attribute elsewhere in the configuration file. For example, the `<exec-deployment>` element could contain `use="myproductiondb"`.	Only if the `use` attribute is used elsewhere in the file.
`db`	Depends on vendor; see the table on page 132.	Names the address string of the destination database.	Yes.*

Attribute	Value	Description	Required?
use	The name of the database set by the name attribute.	If a database is defined in an include file, you can reference it as a destination by including it here. If you reference a database with use, you do not need to specify name or db in the reference because they are already defined in the include file. However, you can optionally set db or any other attribute together with the use attribute, in which case the explicitly set attributes take precedence.	No.
table	Any user-defined table name surrounded by double quotes, for example, "table1".	Names a destination table in db.	Yes.* (not required if you are using user-defined database schemas)
user	Any user name surrounded by double quotes, for example, "user1".	Authorizes a specific database user.	Yes.*
password	Any user-defined password surrounded by double quotes, for example, "w2lYS".	Names the assigned password for user. Note that any password named in a configuration file is not encrypted, and can be read by anyone having access to the configuration file.	Yes.*
timeout	Any positive integer representing the duration of the timeout in seconds, surrounded by double quotes, for example, "4".	How long the client system will attempt to log into the database system before giving up. This tag can also reside in the Destinations section prior to the Database section, in which case it has a different definition. See Item 10 for details.	No.

Attribute	Value	Description	Required?
clear-table	"yes" or "no"	Specifies whether a delta table should be cleared before receiving new data. Useful to set to yes (which is the default) when deleting many workarea files prior to submitting. Set to no if updating extended attributes on existing files prior to submitting. **Note:** This attribute is not supported if you are using user-defined database schemas.	No.
table-view	"yes" or "no"	Specifies whether to create a view automatically during deployment. The default is no. Setting to yes is incompatible with Sybase ASE (but works correctly with Sybase SQL Anywhere and all other supported databases). Setting to yes and using Sybase ASE will result in an aborted deployment. **Note:** This attribute is not supported if you are using user-defined database schemas.	No.

Attribute	Value	Description	Required?
vendor	"microsoft"	Specifies Microsoft SQL Server. Sets a default max-id-length of 128.	Yes.
	"microsoft-inetuna"	Specifies Microsoft SQL Server using an i-net UNA driver. Sets a default max-id-length of 128.	
	"oracle"	Specifies an Oracle database. Sets a default max-id-length of 30.	
	"sybase"	Specifies Sybase SQL Anywhere. Sets a default max-id-length of 128.	
	"ibm"	Specifies IBM DB2. Sets a default max-id-length of 18.	
	"informix"	Specifies an Informix database. Sets a default max-id-length of 18.	
max-id-length	Any positive integer appropriate for an object name length (consult the documentation provided by the database vendor).	Specifies the maximum number of characters in any database object name (for example, column names, table names, and so forth), overriding any defaults set by the vendor attribute.	No.**

* Either here or in the Server section's Database section. See Item 16.

** Not required, but highly recommended. Even if the appropriate value is set by the vendor default, setting it again in max-id-length ensures that the value is explicitly set and easily verified. This also ensures that the value will remain constant should the default value (as set dynamically by DataDeploy) ever change.

User-defined Database Schema <database> Attributes

The DataDeploy 5.0 release introduced the following <database> attributes, many of which support user-defined database schemas:

Attribute	Value	Description	Required?
update-type	base, delta, standalone (Default is standalone.)	Similar to type attribute in <column> element. Value used only if <dbschema> element is present.	Yes.
state-field	Indicates which tuple item will be interpreted as state information.	Specifies the tuple field used for the state value. Similar to state-field attribute of <update> section. Value used only if <dbschema> is present. Added as part of ddcfg_uds.template; all individual configuration files generated by iwsyncdb.ipl –ddgen command inherit the state-field attribute.	No.
enforce-ri	"yes" or "no" (Default is no.)	"Enforce Referential Integrity". If set to yes, CREATE TABLE statements will contain column level and table level constraint clauses, if <foreign-key> elements are defined for <group> elements. Used only if <dbschema> element is present.	No.
ri-constraint-rule	Set to the desired constraint rule clause—for example, "ON DELETE CASCADE". (Default is "".)	"Referential Integrity Constraint Rule". Refer to the Database vendor's manual for CONSTRAINT rule clause. Used only if <dbschema> element is present.	No.

Attribute	Value	Description	Required?
real-update	Enables you specify, independent of the enforce-ri setting, whether updates are performed by deleting existing rows and inserting new ones (default), or by executing a series of UPDATE SQL statements ("real updates").	To configure DataDeploy to perform real updates: • If the enforce-ri attribute is set to YES, do not specify the real-update attribute. • If the enforce-ri attribute is not set or set to No, set the real-update attribute to YES. If you do not want DataDeploy to perform real updates when the enforce-ri attribute is not set or set to YES, set the real-update attribute to No. **Notes**: Do not modify fields that are mapped to primary or foriegn key columns if you set either enforce-ri or real-update to YES.* DATE, DATETIME, TIMESTAMP, CLOB, and BLOB data types are always updated regardless of whether the data has been modified.	No.

Attribute	Value	Description	Required?
delete-tracker	"yes" or "no"	If row updates are required, set to yes when performing standalone deployments if no column in the root group is mapped to path value or no column in any group in <dbschema> is mapped to path value. When this option is set to yes, DataDeploy creates a table called IWDELTRACKER to track primary key values for the root group in order to retrieve, update or delete rows that represent a single data content record. This attribute value is ignored if the root group contains a column mapped to path or if all the groups contain a column mapped to path. If no group contains a column mapped to path and if delete-tracker attribute is not set to yes, deletes are not possible with standalone deployments.	Yes. (Yes only when you need to perform row updates in standalone deployments and if no column in the root group is mapped to path value or no column in any group in <dbschema> is mapped to path value.)
log-level	"1", "2", "3" (Default is 3.)	1: Log errors only 2: Log warnings and errors 3: Log everything	No.
drop-table-prefix	Set to the data content record's category and type—for example, INTERNET_MEDICAL.	Used only when iwdd-op command line option is set to drop-table and <dbschema> element is in the deployment configuration. iwsyncdb.ipl automatically generates value of drop-table-prefix when DAS must process a remove workarea or remove branch event. Refer to drop.cfg.example.	No.

Attribute	Value	Description	Required?
drop-table-suffix	Set to *branch*_STAGING if removing a branch or to drop all base and delta tables for a branch; set to *branch*_WORKAREA_*workarea* if removing a workarea or to drop all delta tables for a workarea.	Used only when iwdd-op command line option is set to drop-table and <dbschema> element is in the deployment configuration. iwsyncdb.ipl automatically generates value of drop-table-suffix when DAS must process a remove workarea or remove branch event. Refer to drop.cfg example.	No.
drop-table	"yes" or "no"	Set to yes if iwdd-op=drop-table command line option is set.	No.
commit-batch-size	positive integer (Default is to deploy all records in a single transaction.)	When a large number of records (data content records or files with extended attributes) are deployed and a failure occurs, by default DataDeploy rolls back the entire deployment. When a positive value is specified for this attribute, DataDeploy commits the deployment after successfully deploying the specified number of records. If an error occurs while processing a batch, only that batch is rolled back. For example, if you are deploying 100 data content records and commit-batch-size is 10, DataDeploy will commit the deployment after the successful deployment of every 10 data content records. If the commit-batch-size attribute is 1, DataDeploy commits every tuple that is successfully deployed.	No.

Attribute	Value	Description	Required?
check-schema	"yes" or "no"	When check-schema is set to yes, DataDeploy detects mismatches caused when a datacapture.cfg (for a data type or the datacapture.cfg in *iw-home*/config) is modified after having deployed data content records or metadata earlier and if new columns are introduced during the modification. The mismatch between existing base and delta tables and the <column> definition in a deployment in a DataDeploy configuration file is detected only if the tables were created in a previous deployment. Do not set this attribute permanently; doing so will cause performance problems because DataDeploy must obtain information from database metadata tables to verify schema consistency. This attribute is applicable to both wide table deployment and deployment with user-defined database schemas.	No.

Attribute	Value	Description	Required?
use-oci	"yes" or "no"	When use-oci attribute is set to yes and the vendor attribute is set to oracle, DataDeploy attempts to connect to the database server using the OCI driver even if no BLOB and/or CLOB data types are involved in deployment. If use-oci attribute is set to yes, the db attribute must be set to the TNS name of the oracle database server.**	No.
schema-helper-cache-size	positive integer (Default is 500.)	Sets cache size. If not set or set to an invalid value, the default is 500.***	No.
schema-helper-cleanup	"yes" or "no"	When set to yes, DataDeploy creates a thread to monitor the cache and to remove least recently used objects from the cache when it becomes full.***	No.
schema-helper-cleanup-interval	positive integer (Default is 2.)	Used only if schema-helper-cleanup is set to yes. Determines how frequently, in minutes, the cache cleanup thread must wake up and check the cache size. If you set schema-helper-cleanup to yes but do not set schema-helper-cleanup-interval, the cleanup thread runs every 2 minutes.***	No.

* If you need to modify such fields, you must clear the value, then save and deploy the DCR. Then insert the new value, save the DCR, and deploy it. Databases report constraint violation errors if child tables reference the field values you are deleting. Therefore, you must delete the corresponding rows in child tables as well. To do that automatically, set the ri-constraint-rule attribute to " ON DELETE CASCADE ". Recreate the rows when you insert the new value for the parent table.

** DataDeploy supports two different JDBC driver protocols when connecting to an Oracle database. The "thin" driver is the default driver used by DataDeploy. When DataDeploy detects CLOB or BLOB data types being used in a deployment, it attempts to connect using the OCI driver; that is, the OCI driver is used only if required under the circumstances. The "thin" driver is the

default because it is Type 4 Pure Java, which does not require any other Oracle client libraries whereas OCI driver depends on Oracle client product.

***While performing transactions (insert, delete, update), DataDeploy converts column values that are in the string format to the actual data type of the column in the target table. For this purpose, it uses the DatabaseMetaData.getColumns() JAVA API to retrieve metadata information about columns in a table. To minimize repeated execution of this Java API, DataDeploy maintains a cache. This attribute in `<database>` allows you to optimize cache for your environment.

Performance Enhancement for Deploying Heavily Nested DCRs

Added in the DataDeploy 5.5.1 release to support a performance enhancement for deploying heavily nested DCRs, the following `<database>` element is used to specify the location of a row map cache file:

Attribute	Value	Description	Required?
`row-map-cache-file`	Full path to a row map cache file.	Row map cache files enable DataDeploy to more quickly deploy DCRs that contain many levels of nested elements.	No.

DataDeploy builds and traverses data tree structures to compute the number of rows it needs to process for tables that map nested elements in DCRs. To speed the process, DataDeploy can be configured to "serialize" (cache in a file) the Java objects produced by such tree traversing, and to read in those objects during actual deployments. When this feature is enabled, DataDeploy builds the serialized Java objects in row maps (contained in a row map cache file) for those `<group>` elements in a `<dbschema>` definition that contain replicants from various nested levels which are mapped to table columns.

Note: When mapping replicants from various nested levels into a `<group>`, you can map multiple replicants from the innermost levels and one of their parent, grand-parent, and so on. You cannot map multiple replicants from a level that is not the innermost nesting level.

When executing deployments that contain DCRs which have replicants from various levels of nesting, DataDeploy checks whether the `row-map-cache-file` is specified. If it is, the serialized Java objects are read from the row map cache file. DataDeploy builds the data structures if an error occurs while reading the row map cache file or if no file is specified in the `row-map-cache-file` attribute.

In DAS mode, DataDeploy stores the most recently used row map cache file in memory.

See "iwsyncdb.ipl Usage" on page 189 for details on how to generate row map files. Note that if <group> elements are modified after a row map file has been generated, you must regenerate that row map file.

db Attribute Syntax

The syntax for the value of the <database> element's db attribute depends on the database vendor. Details are as follows. Syntax and example lines should all be on one line in the DataDeploy configuration file. Line breaks shown here are due to formatting constraints of this document.

Database/Driver	Syntax of db Attribute	Example
Informix	db="//host_name:port/database_name:INFORMIXSERVER=server_name"	db="//sys1.interwoven.com:1357/bank01:INFORMIXSERVER=OL_sys1"
Oracle/JDBC thin	db="*host_name:port:instance_identifier*"	db="host1:1357:db1"
Oracle/JDBC OCI *	db="*database_tnsname*"	db="bank01" (See Oracle documentation for details about configuring TNS names)
Sybase SQL Anywhere	db="*ODBC_data_source_name*"	db="server1"
Sybase ASE	db="*host_name:port/database_name*"	db="sys1.interwoven.com:1357/bank01"
Sybase ASA/jConnect	db="*host_name:port*"	db="sys1.interwoven.com:2638"
Sybase ASA/JDBC-ODBC	db="*ODBC_data_source_name*"	db="server1"

Database/Driver	Syntax of db Attribute	Example
Microsoft SQL Server/ JDBC-ODBC	db="*data_source_name*"	db="bank01" (See Microsoft documentation for details about creating data source names on Windows systems)
Microsoft SQL Server/ i-net UNA	db="*host_name*:*port*?database=*database_name*"	db="localhost:1433?database=datadeploy
IBM DB2 (UDB)	db="//*host_name*:*port*/*database_name*"	db="//host1:1357/db1"

* Used by DataDeploy if Oracle extension data types (for example, CLOB) are used. Requires installation of the OCI client library on the system from which the `iwdd.ipl` command is executed.

The `<database>` subelement also supports the `<stored-procedure>` subelement, which allows you to deploy key-value pairs that are treated as a stored procedure. The `<stored-procedure>` subelement resides in the first nesting level within the `<database>` element, and lets you write a stored procedure using standard SQL syntax as supported by the current database. You can then store the procedure in the database by deploying it as an extended attribute with DataDeploy. Syntax is as follows:

```
<stored-procedure>
      <fieldname prefix="any_prefix_1"/>
      <fieldname prefix="any_prefix_2"/>
      <fieldname prefix="any_prefix_n"/>
</stored-procedure>
```

The value of *any_prefix* can be any case-insensitive character string. DataDeploy will examine each tuple for key-value pairs in which the key name starts with any of the specified prefix values. For each match, the value for that key is treated like a database stored procedure; that is, DataDeploy does not validate the value of the key-value pair for syntax and semantic correctness. Instead, DataDeploy passes the value to the database, the key-value pair is not inserted into the table, and errors (if any) are returned to the user. If creation of stored procedure fails and if the tuple contains non-stored procedure key-value pairs, the entire deployment is aborted.

12. **Rows to update:** The `select` section is where you select database rows to update with data from the current tuple. It is also where you can specify a data type for the deployed data other than the default `VARCHAR 255` (you can also set the data type in Update section; see Item 13, "Update type and related data"). You identify rows by stating one or more matching criteria for column

values in that row. For example, you can select a row whose values in columns named “color” and “size” are respectively “red” and “small.” Column matching criteria are set through the `column` tag. Each `database` section must have exactly one `select` section, and each `select` section must contain at least one `column` tag. Each column tag must contain the following two attributes:

1) `name` or `name-from-field`

2) `value` or `value-from-field`

The column tag can optionally contain the `data-type` and `data-format` attributes.

Syntax is as follows:

Attribute	Description	Value Syntax
`name`	Specifies a column by name.	Text string containing any column name from the table specified by the `database` tag.
`name-from-field`	Specifies a column name by reference to a field in the current tuple.	Any valid tuple field.
`value`	Specifies the literal value to match in the column just named.	Text string containing any table value.
`value-from-field`	Specifies a value to match by reference to a field in the current tuple.	Any valid tuple field.
`data-type`	Specifies the data type for the extended attributes being deployed. If not set, DataDeploy assumes a data type of VARCHAR.	Any data type supported by the database.

Attribute	Description	Value Syntax
data-format	Only required if data-type is set. Specifies the format of the extended attributes being deployed as a date or time. Can be used only under the following conditions: • On an Oracle database server, and when data-type is either DATE, DATETIME, or TIMESTAMP. If you set data-format when any of these conditions do not exist, the setting is ignored. • If data-type is either DATE or DATETIME. **Note:** The format of the data-format value must conform to the specification described for the SimpleDateFormat Java class. For information on the SimpleDateFormat Java class, see "Note on SimpleDateFormat Java Class" on page 136.	Any valid date or time format.
is-replicant	Specifies whether a column is mapped to a replicant field. Use this attribute only with user-defined database schemas.	"yes" or "no"
allows-null	Specifies whether a column can have null values. Use this attribute only with user-defined database schemas.	"yes" or "no"

For example, you would use the following <column> element configuration to deploy the KeyName1 extended attribute values as integers:

```
<column name="ValueCol"
      data-type="INT"
      value-from-field="KeyName1" />
```

Or, to deploy KeyName1 extended attribute values as a date formatted to show *Year-Month-Day Hours:Minutes:Seconds* (assuming an Oracle database):

```
<column name="ValueCol"
      data-type="DATE"
      data-format="yyyy-MM-dd HH:mm:ss"
      value-from-field="KeyName1" />
```

To deploy `KeyName1` extended attribute values as a date formatted to show *`Year-Month-Day`* (assuming an Oracle database), you must change the `data-format` value as shown:

```
<column name="ValueCol"
      data-type="DATE"
      data-format="yyyy-MM-dd"
      value-from-field="KeyName1" />
```

If the `data-type` attribute is not specified in the DataDeploy configuration file, DataDeploy uses `VARCHAR (255)` as the data type. If many columns are created in the table, the total size of each row could easily exceed the maximum row size imposed by the database server. Therefore, it is recommended that you set the `data-type` attribute whenever possible for the columns defined in the `<select>` and `<update>` sections of the DataDeploy configuration file.

Note on SimpleDateFormat Java Class

Use the following table to specify the `data-format` attribute value in the `<column>` element if the data-type attribute is set to `DATE` or `DATETIME`:

Symbol	Meaning	Presentation	Example
G	era designator	(Text)	AD
y	year	(Number)	1996
M	month in year	(Text & Number)	July & 07
d	day in month	(Number)	10
h	hour in am/pm (1-12)	(Number)	12
H	hour in day (0-23)	(Number)	0
m	minute in hour	(Number)	30
s	second in minute	(Number)	55
S	millisecond	(Number)	978
E	day in week	(Text)	Tuesday

Symbol	Meaning	Presentation	Example
D	day in year	(Number)	189
F	day in week in month	(Number)	2 (2nd Wednesday in July)
w	week in year	(Number)	27
W	week in month	(Number)	2
a	am/pm marker	(Text)	PM
k	hour in day (1-24)	(Number)	24
K	hour in am/pm (0-11)	(Number)	0
z	time zone	(Text)	Pacific Standard Time
‘	escape for text	(Delimiter)	
‘’	single quote	(Literal)	‘

Examples using the US locale:

Format Pattern	Result
`"yyyy.MM.dd G 'at' hh:mm:ss z"`	1996.07.10 AD at 15:08:56 PDT
`"EEE, MMM d, ''yy"`	Wed, July 10, '96
`"h:mm a"`	12:08 PM
`"hh 'o''clock' a, zzzz"`	12 o'clock PM, Pacific Daylight Time
`"K:mm a, z"`	0:00 PM, PST
`"yyyyy.MMMMM.dd GGG hh:mm aaa"`	1996.July.10 AD 12:08 PM

13. **Update type and related data:** The `update` section is where you select the type of update, reference table (if applicable), and the table column(s) to update. Update type can be `delta`, `base`, or `standalone` (the default). Type `delta` requires two attributes, `base-table` and `state-field`. The `base-table` attribute names the base table that will be modified after the delta table (named earlier in the `database` section) is updated. The `state-field` attribute names which tuple item will be interpreted as state information. Each `database` section must have exactly one `update` section. The relationship between `update` section settings and the table named earlier in the `database` section’s `table` attribute is as follows:

If the Update section contains this:	And the Database section contains this:	The result is:
`type ="base"`	`table="Table1"`	DataDeploy assumes `Table1` is a base table. Generates a full base table called `Table1` or modifies existing full base table `Table1`.
`type ="base" base-table ="Table2"`	`table="Table1"`	DataDeploy assumes `Table1` is a delta table. Effectively merges rows from delta table `Table1` into base table `Table2`.
`type ="delta" base-table ="Table2"`	`table="Table1"`	DataDeploy assumes `Table1` is a delta table based on the full base table `Table2`. Generates a delta table called `Table1` or modifies existing delta table `Table1`. Does not update `Table2` with delta or any other type of data.

14. **Columns to update:** In the `update` section, you must also select at least one column to update from the row(s) you specified earlier in the `select` section. You select columns by naming matching criteria in `column` tag attributes just as you did in the `select` section. All of the attributes shown in the table in Item 13, "Update type and related data," are supported in the `column` tag as well.

15. **SQL section:** The optional `sql` section lets you create SQL commands that override system defaults and execute automatically during deployment. The `sql` element supports three attributes: `action`, `user-action`, and `type`. Details are as follows:

Attribute	Value	Description
action	create	Lets you define your own SQL CREATE TABLE command for table creation during deployment. Commands set by this attribute override the default DataDeploy schema for creating tables. The default schema is SELECT * FROM *TABLENAME*
	show	Lets you define your own SQL SELECT command for the show-table operation. Commands set by this attribute override the default DataDeploy schema.
	exist	Lets you define database-specific queries to check for the existence of a table. Commands set by this attribute override the default DataDeploy schema.
	tracker-exist	Lets you define database-specific queries to check for the existence of the tracker table. Commands set by this attribute override the default DataDeploy schema.
	tracker-create	Lets you define your own SQL CREATE TABLE command for tracker table creation during deployment. Commands set by this attribute override the default DataDeploy schema.
user-action	*anyname*	Lets you define any arbitrary SQL command(s) for execution during deployment. For example: <sql user-action="showview" type=query> *Arbitrary SQL commands...* </sql> The commands specified here execute only if you set the iwdd-op=do-sql and user-op=*anyname* options on the command line when you invoke DataDeploy. Because the action and user-action attributes are controlled by mutually exclusive command line options, you cannot execute both attributes at the same time (that is, within the same deployment).

Attribute	Value	Description
type	query	If user-action is set, you must also set type. Setting type="query" specifies that user-action will be a query.
	update	If user-action is set, you must also set type. Setting type="update" specifies that user-action will be an update.

Note: It is not necessary for the statements in the <select> and <update> elements to match the table schema in an <sql> element.

16. **Server section:** The server section lets you specify a set of server-specific parameters. A deployment that is expected to run on a server in a three-tier system must have exactly one server section. The bind tag lets you specify where on the server machine the DataDeploy server will listen. Each server section must have exactly one bind section. In a bind section, the port attribute is always required, while the ip attribute is required only if the server machine has more than one available IP address. The optional allowed-hosts element lets you specify which hosts are allowed to connect to the DataDeploy server. If you include an allowed-hosts element, its host subelement must have an addr value in the form of an alphanumeric machine name or an IP address. The optional for-deployment element lets you define several client attributes just as you did in the database section (see Item 11). These attributes are: db, table, user, password, and timeout. If you set these attributes here, they override any settings for the same attributes in the client-side database section. An alternative to including a server section in a client/server configuration file is to have a separate file containing just a server section. This arrangement allows you to separate client and server information into different files, which can reside on different machines.

Sample TeamSite-to-XML Configuration File

The following file configures a typical deployment from TeamSite to an XML file. The `xml-formatted-data` tag has a single attribute, `file`, which specifies the absolute path and file name of the destination file. A `destinations` section can have any number of `xml-formatted-data` elements, or a combination of `xml-formatted-data` and `database` elements. When deploying to an XML file, you can also remap field column tags as shown on page 145.

```
<data-deploy-configuration>
<client>

<deployment name="teamsite-to-xml">
 <source>
    <!-- Pull data tuples from TeamSite EA's -->
    <teamsite-extended-attributes
       options="full"
       area="/default/main/STAGING" >
       <path name="." />
    </teamsite-extended-attributes>
 </source>
 <destinations>
    <xml-formatted-data file="/u/temp/someTable.xml" />
 </destinations>
</deployment>

</client>
</data-deploy-configuration>
```

The following sample file shows the default format of a typical XML destination file:

```
<?xml version="1.0"?>
<xml-tuple-data version="2.0">
 <data-tuple>
    <tuple-field name="path">mydir/f9</tuple-field>
    <tuple-field name="state">Original</tuple-field>
    <tuple-field name="value">small</tuple-field>
    <tuple-field name="key">size</tuple-field>
 </data-tuple>
 <data-tuple>
    <tuple-field name="path">mydir/f9</tuple-field>
    <tuple-field name="state">Original</tuple-field>
    <tuple-field name="value">blue</tuple-field>
    <tuple-field name="key">color</tuple-field>
 </data-tuple>
</xml-tuple-data>
```

Sample Database-to-Database Configuration File

```
<data-deploy-configuration>
<client>
<deployment name="db-to-db">
 <source>
    <!-- Pull data tuples from database -->
    <database db="server"
           user="DBA"
           password="SQL"
           table="staging">
       <fields>
          <field name="path"  column="Path" />
          <field name="key"   column="KeyName" />
          <field name="value" column="Value" />
          <field name="state" column="State" />
       </fields>
    </database>
 </source>
 <destinations>
    <!-- Oracle8 on Unix -->
    <database db="diver:1521:testdb"
           user="scott"
           password="tiger"
           table="someTable">
       <select>
          <column name="Path"
             value-from-field="path"  />
          <column name="KeyName"
             value-from-field="key"   />
       </select>
       <update>
       <!-- Update column 'Value' to contain current EA value, & update column-->
       <!-- 'State' to contain current state. This is a k-v-p specification.  -->
          <column name="Value"
             value-from-field="value" />
          <column name="State"
             value-from-field="state" />
       </update>
    </database>
 </destinations>
</deployment>
</client>
</data-deploy-configuration>
```

In this file, the `field` elements specify which columns in the source database DataDeploy will use when building a tuple for each row. The `select` element chooses rows to update in the destination database. It will choose rows only having unique combinations of the values named in the column subelements (in this case, `path` and `key`). See “Sample TeamSite-to-XML Configuration File” on page 141 for an example of XML destination file format.

Sample Database-to-XML Configuration File: Extracting Data Tuples from a Single Table

The following file configures a deployment from a database to an XML file, including remapped field column tags (as opposed to the default output shown on page 141):

```
<data-deploy-configuration>
<client>

<deployment name="db-to-xml">
 <source>
    <!-- Pull data tuples from databse -->
    <!-- Oracle8 on Unix -->
    <database db="diver:1521:testdb"
           user="scott"
           password="tiger"
           table="tupleTable">
       <fields>
          <field name="path"  column="EPath" />
          <field name="key"   column="EKeyName" />
          <field name="value" column="EValue" />
          <field name="state" column="EState" />
       </fields>
    </database>
 </source>
 <destinations>
    <xml-formatted-data file="/tmp/tupleTable.xml">
    </xml-formatted-data>
 </destinations>
</deployment>

</client>
</data-deploy-configuration>
```

The resulting XML output file is as follows:

```
<?xml version="1.0"?>
<xml-tuple-data version="2.0">
 <data-tuple>
    <tuple-field name="path">mydir/f9</tuple-field>
    <tuple-field name="state">Original</tuple-field>
    <tuple-field name="value">small</tuple-field>
    <tuple-field name="key">size</tuple-field>
 </data-tuple>
 <data-tuple>
    <tuple-field name="path">mydir/f9</tuple-field>
    <tuple-field name="state">Original</tuple-field>
    <tuple-field name="value">blue</tuple-field>
    <tuple-field name="key">color</tuple-field>
 </data-tuple>
</xml-tuple-data>
```

Sample Database-to-XML Configuration File: Filtering

The previous implementation extracts *all* rows from the specified table and creates XML output for the required fields. The following implementation allows you to *filter* the rows that you wish to select from the specified table.

Use the `<database>` element's `where-clause` attribute if you wish to filter table rows. To filter rows, set the `where-clause` attribute to a string value that specifies the row selection criteria.

```
<data-deploy-configuration>
<client>

<deployment name="db-to-xml">
 <source>
    <!--Pull data tuples from database -->
    <!--Oracle8 on Unix -->
    <database db="diver:1521:testdb"
    user="scott"
    password="tiger"
    table="tupleTable"
    vendor="oracle"
    where-clause=" Epath = 'mypath.txt' AND Evalue like 'myvalue%' " >
       <fields>
          <field name="path"column="EPath"/>
          <field name="key"column="EKeyName"/>
          <field name="value"column="EValue"/>
          <field name="state"column="EState"/>
       </fields>
    </database>
 </source>
 <destinations>
    <xml-formatted-data file="/tmp/tupleTable.xml">
    </xml-formatted-data>
 </destinations>
</deployment>

</data-deploy-configuration>
</client>
```

In this example, DataDeploy executes the following query to extract data tuples:

```
SELECT * FROM TUPLETABLE WHERE Epath = 'mypath.txt' AND Evalue like 'myvalue%'
```

Note: You cannot use some of the SQL operators (such as > and <) within the string value for the `where-clause` attribute because doing so may generate XML parser errors. If you must use such operators as part of the `where-clause` attribute value, use the `<db-producer-query>` element, as shown in the next example.

Sample Database-to-XML Configuration File: Extracting Data Tuples from Multiple Tables

The previous examples show database-to-XML deployment implementations that only allow you to extract data tuples from a single table. The following implementation allows you to extract data tuples from multiple tables. This implementation uses the `<db-producer-query>` element as a subelement of `<database>`:

```
<data-deploy-configuration>
<client>

<deployment name="db-to-xml">
 <source>
    <!--Pull data tuples from database -->
    <!--Oracle8 on Unix -->
    <database db="diver:1521:testdb"
    user="scott"
    password="tiger"
    vendor="oracle"
    table="not-used">

       <db-producer-query>
          <![CDATA[
              SELECT A.C1, A.C2, B.C3, B.C4 FROM
                 TABLE1 A, TABLE2 B WHERE
                 A.ID = B.ID AND
                 A.ID = 10
          ]]>
       </db-producer-query>

    </database>
 </source>
<destinations>
 <xml-formatted-data file="/tmp/tupleTable.xml">
 </xml-formatted-data>
</destinations>
</deployment>

</data-deploy-configuration>
</client>
```

Notes:

- To deploy tuples to an XML file, use the `<db-producer-query>` element to specify a complete SQL query statement that DataDeploy should use to extract data tuples. Ensure that the entire SQL select query is within a single CDATA node. If more than one CDATA node is specified under `<db-producer-query>`, DataDeploy will only use the first node.
- Use the `where-clause` attribute for the `<database>` element and the `<db-producer-query>` subelement inside the `<database>` element only if the `<database>` element is a subelement of the `<source>` element.
- If you specify the `<db-producer-query>` subelement, DataDeploy ignores the `<database>` element's `table` and `where-clause` attribute values.
- If you do not specify a `where-clause` attribute value and a `<db-producer-query>` subelement, DataDeploy will select *`all`* rows from the table specified in the `<database>` element's `table` attribute.

Sample XML-to-Database Configuration File

The following file configures a typical deployment from an XML file to a database:

```
<data-deploy-configuration>
<client>

<deployment name="xml-to-db">
 <source>
    <!-- Pull data tuples from XML file -->
    <xml-formatted-data file="/u/iw/wcuan/billTable.xml" >
       <fields>
          <field name="path"  element="path" />
          <field name="key"   element="key" />
          <field name="value" element="value" />
          <field name="state" element="state" />
       </fields>
    </xml-formatted-data>
 </source>
 <destinations>
    <database db="diver:1521:testdb"
          user="scott"
          password="tiger"
          table="TableFromXML">
       <select>
          <column name="Path"
             value-from-field="path"  />
          <column name="KeyName"
             value-from-field="key"   />
       </select>
       <update>
       <!-- Update column 'Value' to contain the current EA value, and -->
       <!-- update column 'State' to contain the current state. This is a -->
       <!-- k-v-p specification. -->
          <column name="Value"
             value-from-field="value" />
          <column name="State"
             value-from-field="state" />
       </update>
    </database>
 </destinations>
</deployment>

</client>
</data-deploy-configuration>
```

In this file, the `field` elements specify which attributes in the source XML file DataDeploy will use when building a tuple for each Path-Key-Value-State item in the file. The `element` attribute can name any valid element; it is not limited to naming just the `path`, `key`, `value`, or `state` elements shown here.

Sample XML-to-XML Configuration File

The following file configures a typical deployment from an XML file to another XML file. This is different than just copying the source file because it includes an in-flow substitution as described in the file comments. You can also include filters when configuring an XML-to-XML deployment, although that feature is not shown here.

```
<data-deploy-configuration>
<client>

<deployment name="xml-to-xml">
 <source>
    <!-- Pull data tuples from XML file -->
    <xml-formatted-data file="/u/iw/wcuan/billTable.xml" >
       <fields>
          <field name="path"  element="path" />
          <field name="key"   element="key" />
          <field name="value" element="value" />
          <field name="state" element="state" />
       </fields>
    </xml-formatted-data>
 </source>
 <substitution>
       <!-- Modify each tuple according to the following  -->
       <!-- match/replace pairs. In this case: any path   -->
       <!-- that contains the string 'WORKAREA/.../' will -->
       <!-- have the string replaced by 'STAGING/'; any   -->
       <!-- path that contains 'EDITION/abcd' will be     -->
       <!-- replace with '/This/Special/Path', and any    -->
       <!-- tuple whose key starts with 'BEFORE' will be  -->
       <!-- changed to begin with 'AFTER'.                -->
       <field name="path"
          match="(.*)/WORKAREA/[^/]+/(.*)"
          replace="$1/STAGING/$2" />
       <field name="path"
          match="EDITION/abcd"
          replace="/This/Special/Path" />
       <field name="key"
          match="^BEFORE(.+)"
          replace="AFTER$1" />
 </substitution>
 <destinations>
    <xml-formatted-data file="/u/temp/someTable.xml" />
```

```
  </destinations>
</deployment>

</client>
</data-deploy-configuration>
```

In this file, the `field` elements specify which attributes in the source XML file DataDeploy will use when building a tuple for each Path-Key-Value-State item in the file.

Starting-State Base Table Configuration File

The following file generates the initial base table BT1 shown in the Starting State diagram on page 40:

```
<data-deploy-configuration>
<client>

<deployment name="staging">
 <source>
    <!-- Pull data tuples from TeamSite EA's -->
    <teamsite-extended-attributes
       options="full"
       area="/default/main/STAGING" >
       <path name="." />
    </teamsite-extended-attributes>
 </source>
 <destinations>
    <!-- Oracle8 on Unix -->
    <database db="diver:1521:testdb"
       user="scott"
       password="tiger"
       table="staging">
       <select>
          <column name="Path"
             value-from-field="path"  />
          <column name="KeyName"
             value-from-field="key"   />
       </select>
       <update type="base"
          state-field="state">
       <!-- Update column 'Value' to contain the current EA value, and -->
       <!-- update column 'State' to contain the current state. This is a -->
       <!-- k-v-p specification. -->
          <column name="Value"
             value-from-field="value" />
          <column name="State"
             value-from-field="state" />
       </update>
    </database>
 </destinations>
</deployment>

</client>
</data-deploy-configuration>
```

Event 1 Configuration File

The following file configures the delta deployment shown in the Event 1 diagram on page 40:

```
<data-deploy-configuration>
<client>

<deployment name="delta">
 <source>
    <teamsite-extended-attributes
       options="differential"
       base-area="/default/main/STAGING"
       area="/default/main/WORKAREA/$workarea" >
       <path name="." />
    </teamsite-extended-attributes>
 </source>
 <destinations>
    <database db="diver:1521:testdb"
       user="scott"
       password="tiger"
       table="Delta_$workarea">
       <select>
          <column name="Path"
             value-from-field="path"  />
          <column name="Key"
             value-from-field="key"   />
       </select>
       <update type="delta"
          base-table="staging"
          state-field="state">
          <column name="Value"
             value-from-field="value" />
          <column name="State"
             value-from-field="state" />
       </update>
    </database>
 </destinations>
</deployment>

</client>
</data-deploy-configuration>
```

Note that this file uses the parameter substitution `$workarea` in the `<database>` section. See “Parameter Substitutions” on page 109 for more information.

Event 2 Configuration File

The following file configures the delta deployment shown in the Event 2 diagram on page 40:

```
<data-deploy-configuration>
<client>

<deployment name="submit">
 <source>
    <teamsite-extended-attributes
       options="differential"
       base-area="/default/main/STAGING"
       area="/default/main/WORKAREA/$workarea">
       <path filelist="/tmp/somefiles" />
    </teamsite-extended-attributes>
 </source>
 <destinations>
    <database db="diver:1521:testdb"
       user="scott"
       password="tiger"
       table="Delta_$workarea">
       <select>
          <column name="Path"
             value-from-field="path"  />
          <column name="Key"
             value-from-field="key"   />
       </select>
       <update type="base"
          base-table="staging"
          state-field="state">
          <column name="Value"
             value-from-field="value" />
          <column name="State"
             value-from-field="state" />
       </update>
    </database>
 </destinations>
</deployment>

</client>
</data-deploy-configuration>
```

Chapter 6

Invoking DataDeploy

This chapter describes how to invoke DataDeploy from the command line, and the conditions under which the DataDeploy daemon runs as a service. You can also use the syntax shown here to invoke DataDeploy through an `iwat` trigger script or an external workflow task as described on page 23.

iwdd.ipl Command

Use the `iwdd.ipl` command to invoke DataDeploy from the command line, in an `iwat` trigger script, or as a workflow task. Usage is as follows. Note that `iwdd.ipl` resides in *`dd-home`*`/bin`.

Usage

```
iwdd.ipl cfg=configfile [deployment=deploymentname][iwdd-op=tableopname]

iwdd.ipl cfg=configfile [deployment=deploymentname][iwdd-op=do-sql] user-
op=anyname mytable=anytable

iwdd.ipl remote-host=hostname [remote-port=portnumber][iwdd-
op=serveropname]
```

Syntax

`iwdd.ipl cfg`	Invokes DataDeploy, and optionally performs table operations.
`cfg=`*`configfile`*	The name of the DataDeploy configuration file, including path name (either absolute or relative to the current directory).
`deployment=`*`deploymentname`*	Invokes DataDeploy as a client. Without this option, DataDeploy is invoked as a server.

deploymentname	The value of the `name` attribute of the `deployment` element.
`iwdd-op=`*tableopname*	Performs the table operation specified by *tableopname*. This is not a standalone option; you can only use it together with `deployment=`*deploymentname*.
tableopname	Displays or deletes tables as follows: `show-table`: Displays an ASCII version of the table named by `table=` in the configuration file's `database` section for the specified deployment. `drop-table`: Deletes the same table from the database. `show-tracker`: Displays an ASCII version of the tracker table. `drop-tracker`: Deletes the tracker table from the database.
`iwdd-op=do-sql`	Performs an SQL operation on the named table.
`user-op=`*anyname*	Performs the user-defined SQL operation defined by `user-action=`*anyname* in the DataDeploy configuration file's `sql` element. See Item 14 in "Sample File Notes" on page 115 section for more information. You must also set `mytable=`*anytable* whenever you set `user-op=`*anyname*.
`iwdd remote-host`	Performs server operations on the server specified in *hostname*.
hostname	The IP address or name of the server host.
`remote-port=`*portnumber*	Specifies the port number on the host. Defaults to `1949` if `remote-port` is not set.
`iwdd-op=`*serveropname*	Performs the server operation specified by *serveropname*. Defaults to `ping-server` if not set.

serveropname	`ping-server`: Returns a standard string to verify the server connection. `stop-server`: Waits for current deployment to complete and then stops the server. All communication with the server is cut off after you issue this command. `kill-server`: Stops the server immediately even if a deployment is running.

Examples

To invoke DataDeploy as a server based on the configuration file `/bin/conf/ddconfig.xml`:

```
iwdd.ipl cfg=/bin/conf/ddconfig.xml
```

To invoke DataDeploy as a client based on the configuration file `/bin/conf/ddconfig.xml` and the deployment named `ea-to-db`:

```
iwdd.ipl cfg=/bin/conf/ddconfig.xml deployment=ea-to-db
```

To delete the tracker table from the database:

```
iwdd.ipl cfg=/bin/conf/ddconfig.xml deployment=ea-to-db iwdd-op=drop-tracker
```

To stop the server on port 1234 of the host `examplehost`:

```
iwdd.ipl remote-host=examplehost remote-port=1234 iwdd-op=stop-server
```

To ping the server on port 1949 of the host `examplehost`:

```
iwdd.ipl remote-host=examplehost
```

Execute the following to invoke DataDeploy as a client to perform the SQL operation `showpaths` on the table `prtable`. In this example:

- The DataDeploy configuration file is `../conf/templating/extranet/pr.cfg`.
- `showpaths` is the value for the `user-op` attribute in the configuration file's `<sql>` element.
- `mytable="prtable"` is a parameter substitution for all occurrences of `$mytable` in the configuration file (see "Parameter Substitutions" on page 109 for more information).

```
iwdd.ipl cfg=../conf/templating/extranet/pr.cfg deployment="dosql" iwdd-
op=do-sql user-op="showpaths" mytable="prtable"
```

Running DataDeploy as a Service

The `Interwoven DataDeploy` service automatically starts the DataDeploy daemon for DAS operation if the `iwsyncdb.cfg` file exists in *`dd-home`*`/conf`. If `iwsyncdb.cfg` does not exist, the `Interwoven DataDeploy` service starts the DataDeploy daemon for 3-tier operation.

Chapter 7

Synchronizing OpenDeploy and DataDeploy

The OpenDeploy and DataDeploy integration enables the deployment of file assets and database assets in one single transactional deployment.

This chapter describes the configuration tasks you must perform to synchronize OpenDeploy with DataDeploy, and how to invoke a deployment after synchronization is complete.

Additional Resources

Refer to the *OpenDeploy Release Notes* for a listing of which releases of DataDeploy are supported by this release of OpenDeploy.

Refer to the `ReadMe` file included with the OpenDeploy software for examples of configuration files and additional information.

Component Location

The solution components reside in the following location on the OpenDeploy host:

od-home`/solutions/ddsync`

Setup

To set up the OpenDeploy-DataDeploy integration, follow these steps:

1. Move the example components to their respective locations described in "Component Descriptions" on page 165.

2. Install the `IWXML.pm` module for use with `iwperl` on the target production server. You can perform this task by copying the following file

   od-home`/solutions/perl/IWXML.pm`

   to any one of the `@INC` paths of `iwperl`. You can view the INC paths by entering the following command at the prompt:

   `iwperl -V`

   You must also modify `ddsync.ipl` on the target production server with the DCR types. See "Component Descriptions" on page 165 for more information.

3. (For internationalization support only) Install the `I18N_utils.pm` module from the following location:

   od-home`/solutions/perl/I18N.pm`

   into `iwperl` through a TeamSite subdirectory under one of the `iwperl @INC` paths. For example, to setup DataDeploy's iwperl, enter the following command at the prompt:

   `cp` *`od-home`*`/solutions/perl/I18N_utils.pm` *`dd-home`*`/iw-perl/lib/perl5/site_perl/5.005/TeamSite`

   If your `iwperl` was installed as part of TeamSite 5.5 or later, this module will already reside there. You must also copy the following file:

   od-home`/solutions/ddsynch/odxmlenc.map`

   to your *od-home* location. This mapping file specifies the encoding you want to be used by the OpenDeploy internal XML log header.

4. Modify the source and location paths as needed:
   – For a full deploy: `oddd_full.xml`
   – For a delta deploy: `oddd_delta.xml`

   You also can take advantage of the OpenDeploy parameter substitution feature for more dynamic specifications.

5. Regenerate by entering one of the following commands at the prompt. Different syntax is used depending on whether wide-table format or user-defined schema support is desired.
   – Wide-table format:

   iwsyncdb.ipl -genloadcfg loaddb.cfg *workarea*

   – User-defined schema:

   iwsyncdb.ipl -genloadcfg *outfile-name* -dumpdir *fullpath-production-dumpdir* -targetdir *fullpath-production-targetdir development-workarea*

6. Manually change `loaddb.cfg` to reflect the following:
   – The path location to `database.xml`
   – Any changes to templating types schemas

Component Descriptions

This section contains the path and name of the OpenDeploy and DataDeploy integration components, and their descriptions. Note that *dd-home* refers to the DataDeploy home directory and *od-home* refers to the OpenDeploy home directory for the base server and receiver software.

- *dd-home*`/bin/ddsync.ipl` — the DNR script. This file is installed on both development and production servers. For user-defined schema support, customization is required. The DCR types must be specified. In `ddsync.ipl`, search for:

  ```
  USER CUSTOMIZATION REQUIRED
  ```

  For internationalization support, your host's local encoding must be specified. This field is also found under the `CUSTOMIZATION` section. The default value is `NONE`.

- *dd-home*/conf/database.xml — the "include" file for the database element. This file is installed on a production server.
- *dd-home*/conf/subxmldb.template — a file used by iwsyncdb.ipl -genloadcfg to generate a loaddb.cfg. This file is installed on the development server to generate loaddb.cfg, which then needs to be relocated to a production server.
- *dd-home*/conf/subxmldb_uds.template — see description for *dd-home*/conf/subxmldb.template above.
- *dd-home*/conf/subxmldb_uds_custom.template — see description for *dd-home*/conf/subxmldb.template above.
- *dd-home*/conf/loaddb.cfg — a generated DataDeploy configuration file used to run DataDeploy to update the database from dump files. This file is generated on the development server, but needs to be relocated to a production server. This file can be regenerated through the following command:

  iwsyncdb.ipl -genloadcfg *output-path*/loaddb.cfg -dumpdir *production-dump-dir* targetdir *production-target-dir area-vpath*

- *dd-home*/conf/tsxml.cfg — a DataDeploy configuration file used to create the XML dump files from TeamSite. This file is installed on the development server.
  - Using wide-table format — copy from */tsxml.cfg.example-wide
  - Using user-defined-schema — copy from */tsxml.cfg.example-uds
- *od-home*/conf/odrcvr.xml — the configuration file for an OpenDeploy host with the receiver software installed. This type of OpenDeploy host can only receive files. This file is installed on a production server.
- *od-home*/conf/oddd_full.xml — an OpenDeploy source host configuration file. This file is installed on the development server.
- *od-home*/conf/oddd_delta.xml — an OpenDeploy source host configuration file. This file is installed on development machine.
- *od-home*/odxmlenc.map — an OpenDeploy mapping file. This mapping file specifies encoding to use in the OpenDeploy internal XML log.

Usage

This section describes how these integration tools are used.

- On a development server, run the following command at the prompt:

  `iwodstart oddd_full` *or*

  `iwodstart oddd_delta`

  If you want to use parameter substitution, enter the following command at the prompt:

  `iwodstart oddd_full -k "var1=value1"`

  where the *$var1* parameter is present somewhere in the `oddd_full.xml` configuration file.

- On a production server, start up the OpenDeploy receiver services or daemons.

How the Integration Works

This section describes the processes that OpenDeploy and DataDeploy perform to complete the integration.

1. OpenDeploy runs the specified deployment transactionally.
2. A pre-deploy Deploy and Run script is kicked off to invoke DataDeploy.
   - The script (`ddsync.ipl`) is a wrapper that invokes DataDeploy to extract TeamSite DCR or TeamSite metadata into XML dump files.
   - The configuration files for the DataDeploy invocation have been prepped to dump out the DCR types and metadata of a specified area.
3. OpenDeploy transfers its normal set of files plus the XML dump files.

4. A post-deploy Deploy and Run script is kicked off to invoke DataDeploy:
   - The `ddsync.ipl` script invokes DataDeploy to import into the configured database all the actual DCR files. No transient XML dump file is used here.
   - The config files for the DataDeploy invocation have been prepped to read a series of generated file lists of the DCR types. Metadata is the exception and still follows the wide-table scheme. The generated file lists are derived from parsing the OpenDeploy internal XML log. For internationalization, these generated file lists are in the UTF-8 encoding for DataDeploy usage.
5. Both file assets and database assets should now be deployed. If any part of the deploy should fail, the entire deployment should be restored back to previous state. For example:
   - If the DataDeploy component fails, the OpenDeploy deployed files will be reverted.
   - If the OpenDeploy component fails, the DataDeploy component will not be invoked.

ddsync.ipl Usage

Here are the different ways `ddsync.ipl` can be used:

Wide-Table Format:

`ddsync.ipl` *`area_top`* *`dump_dir`* `dump full` *`area`*

`ddsync.ipl` *`area_top`* *`dump_dir`* `dump differential` *`area basearea`*

`ddsync.ipl` *`area_top`* *`dump_dir`* `load full`

`ddsync.ipl` *`area_top`* *`dump_dir`* `load differential`

User-Defined Schema:

```
ddsync.ipl area_top dump_dir dumpea full area

ddsync.ipl area_top dump_dir dumpea differential area basearea

ddsync.ipl area_top dump_dir loaduds full

ddsync.ipl area_top dump_dir loaduds differential
```

area_top	The absolute path to top of area directory.
dump_dir	The relative path to dump directory in area.
dump	Dump TeamSite metadata to file.
load	Load database from dump file.
load_uds	Load database from DCR filelist.
full	The entire area traversal.
differential	Comparing between two areas.
basearea	If `differential` is used, the *basearea* is the previous area.
area	If `differential` is used, the *area* is the current area.

The `ddsync.ipl` usage also creates the following log file:

dd-log-home`/ddsync_dump_load.log`

Notes

- DCRs targeted for deployment must be located within the `templatedata` directory structure on the source system and deployed to the `templatedata` directory structure on the recipient system.
- Run only directory comparison deployments. TeamSite-based and filelist modes are not supported at this time.
- OpenDeploy-DataDeploy integration is most suitable for deployment of editions.
- Because OpenDeploy uses remote file system comparison and DataDeploy is either full or differential, the consistency of the files and their respective database data is not tightly maintained. The deployment workflow process must be tightly adhered to. For example, the first development-to-production deployment of an edition will deploy all files and all database assets. Subsequent deployments will be directory comparisons between a subsequent edition and the production server and tuple-diffs between current edition and the previous edition. Injection of new files or database content on any point outside of the edition deployment process will create data inconsistencies.
- Be sure to change all references to `/local/`*`iw-home`* in `ddsync.ipl` and `oddd_send.cfg` to reflect the location of the actual OpenDeploy home directory. The `oddd_send.cfg` and `oddd_receive.cfg` files are OpenDeploy sender and receiver host configuration files, and as such require specific configuration for both types of hosts. If these configuration files already exist on systems targeted for the integration, the sample configuration files should be integrated into the production systems.

Chapter 8

Automating Deployment with DAS

This chapter describes how to configure and use the Database Auto-Synchronization (DAS) module. It contains the following sections:

- Overview
- Software Re quirements
- DAS Program and Configuration Files
- Configuring DAS

Overview

The DAS module is bundled with DataDeploy. After you configure DAS, it automatically deploys data content records (DCRs) or extended attributes (metadata) to a database whenever a TeamSite user:

- Creates, changes, or deletes a data content record through the TeamSite Templating GUI.
- Creates, changes, or deletes a file, TeamSite area, or branch containing extended attributes through the command line.
- Creates, changes, or deletes a file, TeamSite area, or branch containing extended attributes through the TeamSite file system interface.

The following table summarizes DAS deployment support for various types of tuples:

Source	Narrow Tuples	WideTuples	Tuples Mapped to User-Defined Database Schemas
Data content records	Not applicable	Supported	Supported
Extended attributes	Not supported	Supported	Not supported

DAS deploys by running DataDeploy as a daemon, and by using various TeamSite events as triggers to initiate deployment. You cannot use DAS when using a three-tier architecture, as described on page 18 and on page 175. The following sections describe how to configure and run DAS.

DAS , The Event Server, and Internationalization

The Event Server is a component of TeamSite that, among other advantages, enables you to filter the events that DAS receives. The ability to filter events greatly improves the scalability and reliability of DAS. For details about using DAS with the Event Server, see Appendix C, "Event Server."

If you operate DAS in a non-US English environment, you must use DAS with the Event Server. For details about Internationalization, see Appendix D, "Internationalization."

Software Requirements

To use DAS, you must first install and configure the following Interwoven products:

- TeamSite (see the *TeamSite Administration Guide*)
- DataDeploy (described earlier in this manual)

Note: If TeamSite Templating is not installed and configured for the area being deployed, only metadata is sent. TeamSite also refers to metadata as extended attributes.

DAS Program and Configuration Files

The files listed in the following table control the operation of DAS. These files are installed automatically when you install DataDeploy, with the exception of `iw.cfg`, `daemon.cfg` and `ddcfg.template`.

Note: `iw.cfg` is installed with TeamSite. In order to avoid overwriting existing `daemon.cfg` and `ddcfg.template` files, the DataDeploy installation procedure creates `daemon.cfg.example` and `ddcfg.template.example` files. If you are performing a first-time installation, rename these files by removing the `.example` suffix.

See the sections following the table for configuration instructions.

File	Location	Description
daemon.cfg	*dd-home*/conf	Configuration file used by the DataDeploy daemon for start-up. You do not need to configure daemon.cfg before running DAS. However, you can optionally add <allowed-hosts> and <bind> tags to daemon.cfg to further control access to the database server. See Item 16 in "Sample File Notes"for more information.
ddcfg.template	*dd-home*/conf	Template DataDeploy configuration file used by ddgen.ipl as a basis for creating all the working DataDeploy configuration files for data types. You must configure ddcfg.template as described in "Editing ddcfg.template and drop.cfg" on page 175 before running DAS. **Note**: The <database> definitions in DataDeploy configuration files are now located in a single file, database.xml.
ddcfg_uds.template	*dd-home*/conf	Template DataDeploy configuration file used to deploy data to user-defined schemas. **Note**: The <database> definitions in Datadeploy configuration files are now located in a single file, database.xml.
ddcfg_uds.custom.template	*dd-home*/conf	Template DataDeploy configuration file used to deploy data from custom DCRs to user-defined schemas. **Note**: The <database> definitions in Datadeploy configuration files are now located in a single file, database.xml.
ddgen.ipl	*dd-home*/bin	DataDeploy configuration file generator. You do not need to configure ddgen.ipl before running DAS. Contact Interwoven before attempting to modify ddgen.ipl.
drop.cfg	*dd-home*/conf	Utility configuration file used by the DataDeploy daemon when dropping tables. You must configure drop.cfg as described in "Editing ddcfg.template and drop.cfg" on page 175 before running DAS.

File	Location	Description
`iwsyncdb.cfg`	*dd-home*`/conf`	Configuration file for `iwsyncdb.ipl`. Controls name and port number for the DataDeploy daemon host. Also controls DataDeploy event logging. See "Editing iwsyncdb.cfg" on page 175 for more information.
`iwsyncdb.ipl`	*dd-home*`/bin`	TeamSite event trigger program and CLT. You do not need to configure `iwsyncdb.ipl` before running DAS. Contact Interwoven before attempting to modify `iwsyncdb.ipl`.
`iw.cfg`	`/etc (on Solaris)` *iw-home*`/etc (on Windows)`	Controls whether renaming, moving, or deleting files will trigger deployment. See "Editing iw.cfg" on page 176 for more information.

Configuring DAS

You must perform the following steps to configure DAS following a DataDeploy installation:

- Edit the main TeamSite configuration file, `iw.cfg`.
- Edit configuration files that are specific to DataDeploy.
- Run the main DataDeploy configuration script, `iwsyncdb.ipl`.

The following subsections describe these steps in detail.

Note: If you wish to deploy to a Microsoft SQL Server database using the JDBC-ODBC bridge, you must also create a Data Source Name (DSN) on your Windows machine for the Microsoft SQL Server database. (See your operating system documentation for details about creating a DSN.) In addition to creating a DSN, ensure that the value of the `<database>` element's `db` attribute in DataDeploy configuration files is equal to the DSN.

Editing DataDeploy Configuration Files

This section describes how to configure the `ddcfg.template`, `drop.cfg`, and `iwsyncdb.ipl` files with your site-specific information.

Editing ddcfg.template and drop.cfg

You must set the following attributes in each <database> element in ddcfg.template and drop.cfg as described item 11 on page 121 in the "Sample File Notes" section.

- db
- user
- password
- vendor

For example, the following settings in ddcfg.template cause DataDeploy to connect to the Sybase SQL Anywhere database server as marketing (using the password al450) and deploy data to the marketingdb database on port 1521 of the server dbserver1:

```
<database   name     = "myproductiondb"
            db       = "localhost:2638"
            user     = "marketing"
            password = "al450"
            vendor   = "SYBASE" >
```

Note: Do not edit the name attribute.

You must configure these settings within each occurrence of the <database> element. For example, if the <database> element occurs four times in ddcfg.template, you must configure these settings identically in all four locations. The same requirement applies to drop.cfg. You must reconfigure these settings in both files whenever you change a database, user, or password.

Editing iwsyncdb.cfg

The *dd-home*/conf/iwsyncdb.cfg file controls the following DataDeploy parameters:

Parameter	Setting in iwsyncdb.cfg
DataDeploy daemon host port number (host name is set automatically to the local host name of the TeamSite server).	Set daemon_port=*number*. For example, to set the port number to 3456, enter daemon_port=3456
Logging of DataDeploy events.	Set suppress_log=yes to disable logging. Set to no to enable logging.

Editing iw.cfg

You must edit the `[iwserver]` section of `/etc/iw.cfg` as follows to support DAS recognition of TeamSite events. Once configured, DAS will support these events whether they are initiated from the TeamSite GUI, the TeamSite file system interface, or the command line (through standard operating system commands or TeamSite command-line tools such as `iwextattr`).

TeamSite Event	Setting in iw.cfg
Delete a file.	`log_syncdestroy=yes`
Revert a file containing extended attributes to an earlier version.	`log_syncrevert=yes`
Rename/move a file.	`log_renamefse=yes`
Move a file.	`log_renamefse=yes`
Set extended attributes on a file.	`log_setea=no`
Delete extended attributes from a file.	`log_deleteea=no`

Running iwsyncdb.ipl

This section describes how to run the `iwsyncdb.ipl` script, which performs the following activities:

- Generates DataDeploy configuration files for use by the DataDeploy daemon.
- Submits the generated DataDeploy configuration files to the staging area and publishes an edition based on the updated staging area.
- Establishes TeamSite events as triggers for automatic data deployment.
- Starts the DataDeploy daemon.
- Creates initial base and delta tables in the destination database for the updated TeamSite areas.

The following subsections and diagrams explain these activities in detail.

Starting iwsyncdb.ipl

Enter the following command to start the `iwsyncdb.ipl` script:
`dd-home``/bin/iwsyncdb.ipl -initial` *`workarea_vpath`*

For *`workarea_vpath`*, specify the full *`vpath`* to the TeamSite Templating workarea that was set up earlier as described in Step 1 of “Copying the Example Directory Structure” in the “Initial

Configuration" chapter of the *TeamSite Templating Developer's Guide*. For example, you would enter the following if the TeamSite Templating subbranch `b1` and workarea `w1` are on the `default/main` branch, and *`dd-home`* is `/usr/iw-home/datadeploy`:

```
/usr/iw-home/datadeploy/bin/iwsyncdb.ipl -initial /default/main/b1/WORKAREA/w1
```

iwsyncdb.ipl Activities

The following figures show the activities that take place when `iwsyncdb.ipl` runs. Activities are grouped as follows:

- Figure 1: Generation of DataDeploy Configuration Files
- Figure 2: Other DAS Setup Activities

All of the activities shown in Figures 1 and 2 occur when you enter `iwsyncdb.ipl` on the command line. You do not need to execute `iwsyncdb.ipl` a second time to initiate the activities shown in Figure 2.

Generation of DataDeploy Configuration Files

The following figure shows how DataDeploy configuration files are generated, submitted, and published when the `iwsyncdb.ipl` script runs. See the diagram key following the diagram for details about each step. Ensure that you read the note that follows the diagram key; this note describes what to do if `iwsyncdb.ipl` generates DataDeploy configuration files but fails to connect to your database.

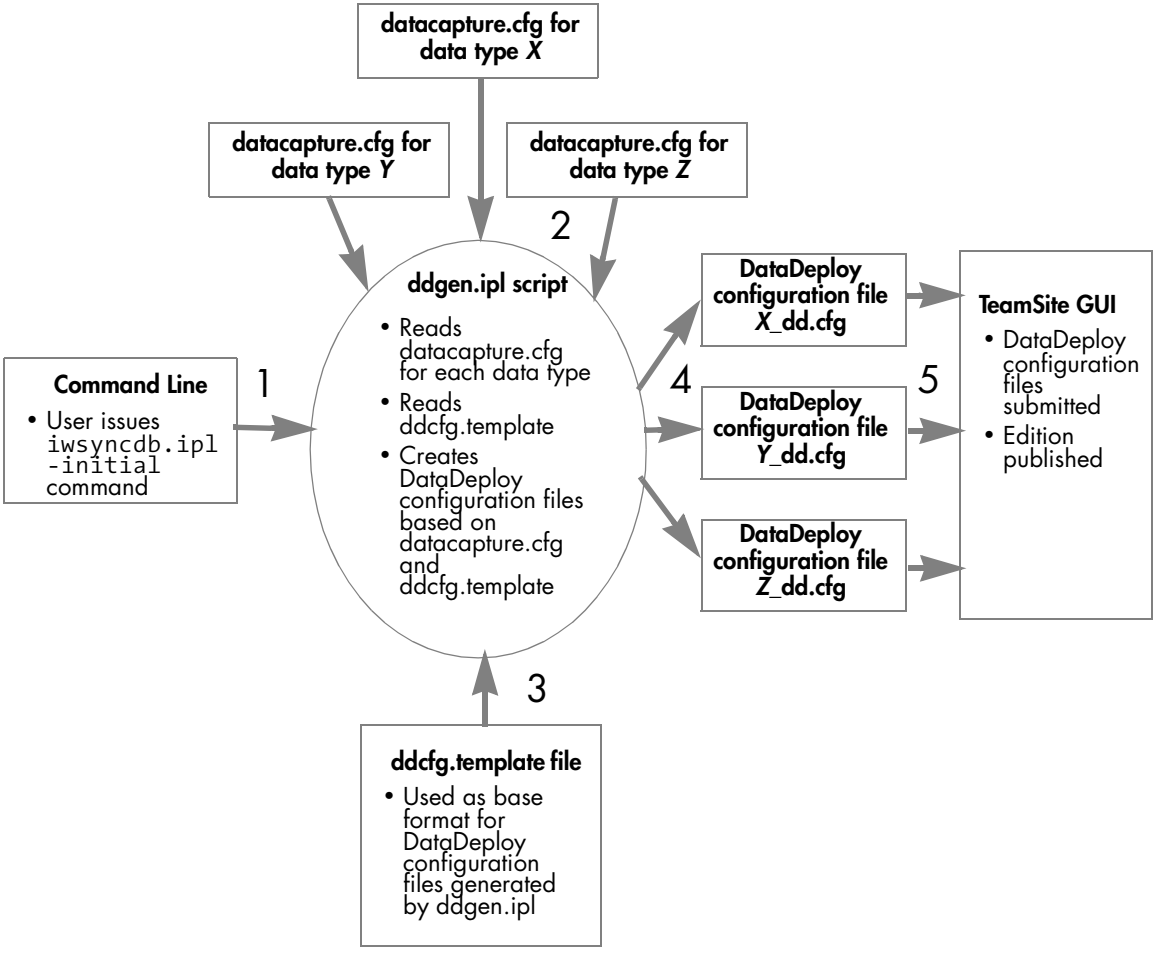


Figure 1: Generation of DataDeploy Configuration Files

Figure 1 Key

1. The `iwsyncdb.ipl -initial` command is executed from the command line as described in "Starting iwsyncdb.ipl" on page 176. The `iwsyncdb.ipl` script starts the `ddgen.ipl` script.
2. The `ddgen.ipl` script reads the TeamSite `datacapture.cfg` file for each data type that exists in *`workarea_vpath`* specified in Step 1. For example, if the TeamSite Templating directory in *`workarea_vpath`* contains the data types *`X`*, *`Y`*, and *`Z`*, `ddgen.ipl` reads the `datacapture.cfg` file corresponding to each data type.

   See "Configuring the Example Templating Environment" in the *TeamSite Templating Developer's Guide* for details about data types and the `datacapture.cfg` file.
3. The `ddgen.ipl` script uses `ddcfg.template` as the base format of the DataDeploy configuration files that it will generate for each data type.
4. Based on `ddcfg.template` and the `datacapture.cfg` files for each data type, `ddgen.ipl` creates DataDeploy configuration files for each data type. Continuing with the example from Step 2, `ddgen.ipl` creates DataDeploy configuration files *`X`*`_dd.cfg`, *`Y`*`_dd.cfg`, and *`Z`*`_dd.cfg`. These configuration files configure a TeamSite-to-database deployment similar to the one described in "Sample TeamSite-to-Database Configuration File" on page 109. The `mdc_dd.cfg` file is also created to ensure that DataDeploy remains synchronized with other TeamSite features such as metadata capture and metadata search.
5. The newly generated DataDeploy configuration files are submitted to the staging area, and an edition based on the updated staging area is published.

Note: If `iwsycndb.ipl` succeeds in generating DataDeploy configuration files for the data types (*`data_type`*`_dd.cfg`), but fails to connect to your database, the following problem will occur: Changes to the `vendor` attribute or to other `<database>` attributes in the `ddcfg.template` file will not be propagated to the *`data_type`*`_dd.cfg` files.

If this occurs, you must edit all *`data_type`*`_dd.cfg` files or you must use the `-force` option of `iwsyncdb.ipl` to overwrite the *`data_type`*`_dd.cfg` files.

Other DAS Setup Activities

The following figure shows how the remaining DAS setup activities take place when the `iwsyncdb.ipl` script runs. See the diagram key following the diagram for details about each step.

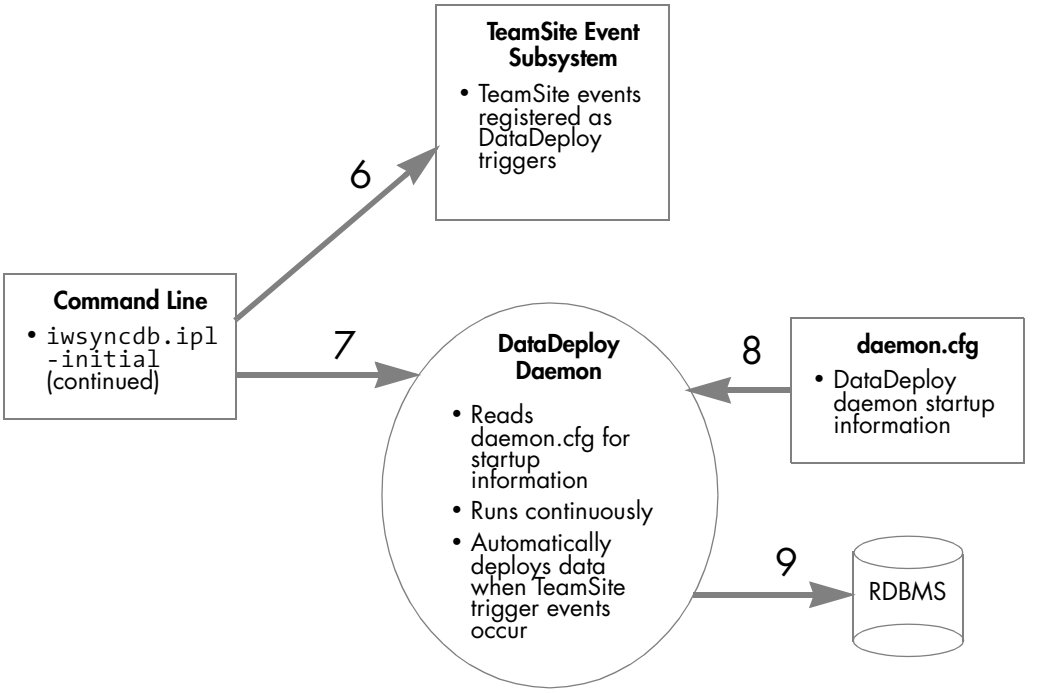


Figure 2: Other DAS Setup Activities

Figure 2 Key

6. The `iwsyncdb.ipl` script registers a default set of TeamSite events as triggers that will automatically initiate deployment. See "TeamSite Event Triggers" on page 185 for details about which events are registered as triggers.

7. The `iwsyncdb.ipl` script starts the DataDeploy daemon.

8. The DataDeploy daemon reads the `daemon.cfg` file, which contains additional daemon startup information. The daemon finishes its startup, and runs continuously until DAS is disabled as described in "Disabling DAS" on page 188.

9. The DataDeploy daemon creates the following in the destination database:
   – Initial wide base tables for the branch.
   – Initial delta tables and views for the workarea.

DAS is now configured and ready for use. The only time you need to repeat any configuration step is when you enable a different database, user, or password. If you add new templating branches, workareas, or files through the TeamSite GUI, DAS automatically generates the necessary DataDeploy configuration files and initial tables.

Using DAS

After DAS is configured, it is transparent to TeamSite templating end users. Therefore, there are no additional tasks that an end user must perform to use DAS. The following diagram shows how DAS automatically updates the necessary tables when a TeamSite trigger event occurs. See the diagram key following the diagram for details about each step.

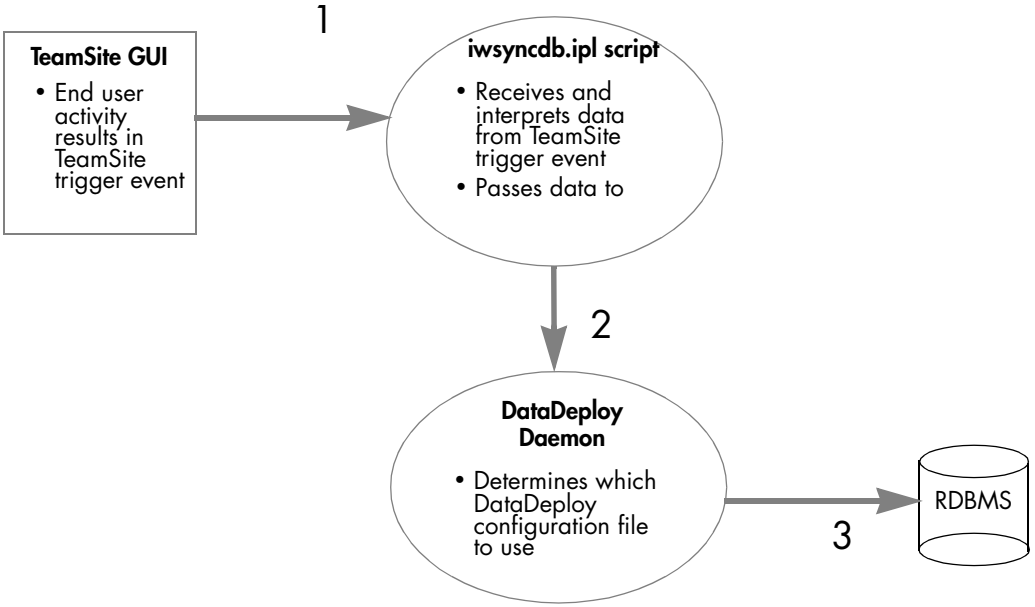


Figure 3: Using DAS

Figure 3 Key

1. TeamSite Templating end-user activity (that is, any activity shown in "TeamSite Event Triggers" on page 185) results in a TeamSite event trigger. The event trigger starts the `iwsyncdb.ipl` script and sends the changed data to the script.

2. The `iwsyncdb.ipl` script sends the data content record to the DataDeploy daemon. The daemon determines which DataDeploy configuration file(s) to use for the deployment. For TeamSite events (for example, **Create Branch**) that are not specific to a single file, the daemon uses the `templating.cfg` file to determine which data types (and therefore which DataDeploy configuration files) are affected by the TeamSite event. For example, in the case of a **Create Branch** TeamSite event, the daemon reads `templating.cfg` to determine which data types exist in the branch. The daemon then uses the DataDeploy configuration files for each affected data type when deploying the new data to the database.

   For events that are file-specific (for example, renaming a file), the daemon uses the information from the TeamSite event information module to determine which file is affected and which DataDeploy configuration file to use.

3. The daemon uses the appropriate DataDeploy configuration file(s) to update the affected base and delta tables in the database. The following section, "Table Update Details" describes these updates.

Table Update Details

This section describes how the base and delta tables described in the preceding section change as data is deployed. This example shows a hypothetical update to a data content record. In this example:

- The data category is `internet`.
- The data type is `pr` (press release).
- The branch is `b1`.
- The workarea is `w1`.

Specifying How Tables are Updated

You can specify how DataDeploy updates tables. DataDeploy can update tables in the following ways:

- By deleting existing rows and inserting new ones (default).

- By executing a series of UPDATE SQL statements. That method is refered to as “real updates.”

Two attributes in the <database> element in DataDeploy configuration files enable you to specify which kind of update you want DataDeploy to perform:

- enforce-ri
- real-update

See page 121, note 11 “Database section”, in the sample configuration file notes for details about how to specify those attributes.

Notes: Do not modify fields that are mapped to key columns when you use real updates, because relational databases do not allow the modification of values in DCRs that are mapped to key columns (primary or foreign).

If you need to modify such fields, you must clear the value, then save and deploy the DCR. Then insert the new value, save the DCR, and deploy it. Databases report constraint violation errors if child tables reference the field values you are deleting. Therefore, you must also delete the corresponding rows in child tables. To do that automatically, set the ri-constraint-rule attribute to " ON DELETE CASCADE ". Recreate the rows when you insert the new value for the parent table.

DATE, DATETIME, TIMESTAMP, CLOB, and BLOB data types are always updated regardless of whether the data has been modified.

Table Naming Conventions

Base and delta tables use the following naming convention:

datacategory_datatype__branchname_areaname

This naming convention includes using double underscores (_+_) between *datacategory_datatype* and *branchname_areaname*.

For example:

`internet_pr__b1_staging` (a base table for the staging area on the `default/main/b1` branch)

`internet_pr__b1_workarea_w1` (a delta table for the workarea `w1` on the `default/main/b1` branch)

Table Update Examples

When the initial wide base table is created as described in Figure 2, Step 9, it contains a Path column, a State column, and columns for each *item* in the data content record. In this starting state, the table does not yet contain any values. Assume that the first three items are `PressDate`, `Headline`, and `Picture`. The resultant wide table will look like this:

Path	State	PressDate	Headline	Picture	...

Wide Base Table for Staging Area (Starting State)

When the initial delta table is created, it contains the same columns as the initial base table, in addition to values for each item:

Path	State	PressDate	Headline	Picture	...
mypath	New	11/17/99	New Candidate Enters Race	cand.gif	...

Delta Table for Workarea (Starting State)

When the data content record is submitted, its delta table values are transferred to the base table, and its own cells are cleared, as shown in the following two tables:

Path	State	PressDate	Headline	Picture	...
mypath	New	11/17/99	New Candidate Enters Race	cand.gif	...

Base Table for Staging Area (Ending State)

Path	State	PressDate	Headline	Picture	...

Delta Table for Workarea (Ending State)

TeamSite Event Triggers

DAS interprets the following TeamSite events as deployment triggers. The event can be initiated from the TeamSite GUI, the TeamSite file system interface, or the command line. Whenever one of these events occurs, the delta and base tables are updated as shown:

TeamSite Event	Delta Table Action	Base Table Action
Create Branch	None.	Build empty base tables.
Create Workarea	Build delta tables.	None.
Delete Branch	Delete delta tables.	Delete base tables.
Delete Workarea	Delete delta tables.	None.
Modify data content record	Update or insert a new row.	None.
Add data content record	Insert a new row.	None.
Delete data content record	Insert or update the Not Present row.	None.

TeamSite Event	Delta Table Action	Base Table Action
Submit modified data content record	1. The Previous Staging row is propagated to all workareas except the submitting workarea. 2. Delete Previous Staging row from submitting workarea.	Update the Staging row.
Submit added data content record	1. The Placeholder row marked NOT-PRESENT is propagated to all workareas except the submitting workarea. 2. Delete the Placeholder row from the submitting workarea.	Update the Staging row.
Submit deleted data content record	1. The previous Staging row is propagated to all workareas except the submitting workarea. 2. Delete the previous Staging row from the submitting workarea.	Update the Staging row.
Get Latest (workarea)	Rebuild the delta tables.	None.
Copy To (any area)	Rebuild the delta tables.	None.
Rename Workarea	1. Delete the old delta tables. 2. Regenerate new delta tables.	None.
Rename Branch	None.	1. Delete the old base tables. 2. Regenerate new base tables.
Rename Directory	Regenerate new delta tables.	None.
Rename File	1. Delete the row for the old file name. 2. Add a row for the new file name.	None.
Move File	1. Delete the row for the old file name. 2. Add a row for the new file name.	None.

TeamSite Event	Delta Table Action	Base Table Action
Delete File	If a row for the file exists in the base table, the row in the delta table is marked NOT-PRESENT. If no row existed in the base table, the row in the delta table is deleted.	None.
Set extended attributes	Insert or update the row.	None.
Delete extended attributes	In a wide table, rows are updated. In a narrow table, the row is deleted or marked `NOT-PRESENT`.	None.
Revert	Use the data from the earlier version of the file (selected in the TeamSite graphical user interface) to update or insert a new row.	None.

Logging DAS Activities

By default, all DAS activities are recorded in *dd-home*`/iwevents.log`. If updates to this file degrade system performance, you can turn off logging for any of the TeamSite events shown in the table in "Editing iw.cfg" on page 176. Use the following event names when you disable logging:

- `RenameFSE`
- `SyncDestroy`
- `SetEA`
- `DeleteEA`
- `SyncRevert`

For example, to prevent **Rename** events from being recorded, set the following in `iw.cfg`:

```
iwevents_exclude="RenameFSE"
```

You can also use regular expressions with the following syntax to further control event logging:

```
renamefse_filter="REGEX"
```

For example, to specify that only **Rename** events occurring in the workarea `bill` are logged:

```
[iwserver]
renamefse_filter="/default/main/WORKAREA/bill"
```

This entry sets regular expressions, one of which must match the event line (as seen in `iwevents.log`) in order for an event to be logged. If these are empty or absent, all corresponding events are logged.

Disabling DAS

Issue the following command to remove the TeamSite event trigger scripts and stop the DataDeploy daemon:

dd-home`/bin/iwsyncdb.ipl -uninstall`

To re-enable DAS after it has been disabled, issue the following command:

dd-home`/bin/iwsyncdb.ipl -install`

Note that you do not need to regenerate the `datacapture.cfg` files that were generated earlier during DAS configuration. See the next section, "iwsyncdb.ipl Usage," for more information about the `iwsyncdb.ipl` command.

iwsyncdb.ipl Usage

Usage

```
iwsyncdb.ipl  [
                -h | -install | -uninstall | -iwat | -iwrmat |
                -startddd | -stopddd | -ddgen vpath [dcr-type] [-force] |
                -dbschemagen vpath [dcr-type] [-force] |
                -initial vpath [dcr-type] | -mb | -mdcddgen [-force] |
                -mdcdbschemagen [-force] | -resyncbr vpath [dcr-type] |
                -resyncwa vpath [dcr-type] | -rmbr vpath [dcr-type] |
                -rmwa vpath [dcr-type] | -rowmapgen configfile
                 deployment outputfile | -showbase vpath [dcr-type] |
                -showdelta vpath [dcr-type] |-showtracker |
                -synctracker vpath| -validate vpath [dcr-type] |
                -validate dbschema_file_name
                ]
```

Option	Description
`-install`	Installs the database synchronization triggers and starts the DataDeploy daemon.
`-uninstall`	Removes the TeamSite event trigger scripts and stops the DataDeploy daemon.
`-iwat`	Registers the `iwsyncdb` trigger scripts.
`-iwrmat`	Unregisters the `iwsyncdb` trigger scripts.
`-startddd`	Starts the DataDeploy daemon.
`-stopddd`	Stops the DataDeploy daemon.
`-ddgen vpath [dcr-type]`	Generates DataDeploy configuration files for data types configured in *`iw-home`*`/local/config/templating.cfg` under the specified workarea *`vpath`*. The `-force` option overwrites any existing configuration files. The optional *`dcr-type`* setting specifies creation of a configuration file for a single data type (rather than all data types in *`vpath`*).

Option	Description
`-dbschemagen vpath [dcr-type]`	Generates DataDeploy `dbschema.cfg` files for data types configured in *`iw-home`*`/local/config/templating.cfg` under the specified workarea *`vpath`*. The `-force` option overwrites any existing configuration files. The optional *`dcr-type`* setting specifies creation of a configuration file for a single data type (rather than all data types in *`vpath`*).
`-initial vpath [dcr-type]`	Generates the initial base and delta tables for the first template-enabled workarea *`vpath`*. The optional *`dcr-type`* setting specifies creation of tables for a single data type (rather than all data types in *`vpath`*).
`-mb`	Encodes all configuration files it generates as UTF-8,and enables all other `iwsyncdb.ipl` options that use Vpath arguments to be processed by a Java equivalent of `iwsyncdb.ipl`. DataDeploy supports an `-mb` options when running `iwsyncdb.ipl`.
`-mdcddgen`	Generates the DataDeploy configuration file `mdc_dd.cfg` (based on *`iw-home`*`/local/config/datacapture.cfg`) for use by the metadata capture subsystem. The `-force` option overwrites any existing configuration files.
`-mdcdbschemagen`	Generates the DataDeploy database schema file *`iw-home`*`/local/config/dbschema.cfg` (based on *`iw-home`*`/local/config/datacapture.cfg`) for use by the metadata capture subsystem. The `-force` option overwrites any existing `dbschema.cfg` files.
`-resyncbr vpath [dcr-type]`	Regenerates the base tables for the branch named by *`vpath`* and the delta tables for the underlying workareas. The optional *`dcr-type`* setting specifies resynchronization of tables for a single data type (rather than all data types in *`vpath`*). Run `iwfreeze` to freeze the backing store before regenerating.
`-resyncwa vpath [dcr-type]`	Regenerates the delta tables for the workarea *`vpath`*. The optional *`dcr-type`* setting specifies resynchronization of tables for a single data type (rather than all data types in *`vpath`*). Run `iwfreeze` to freeze the backing store before regenerating.

Option	Description
`-rmbr vpath [dcr-type]`	Deletes the base tables for the branch named by *`vpath`*. The optional *`dcr-type`* setting specifies deleting the base table for a single data type (rather than all data types in *`vpath`*).
`-rmwa vpath [dcr-type]`	Deletes the delta tables for the workarea named by *`vpath`*. The optional *`dcr-type`* setting specifies deleting tables for a single data type (rather than all data types in *`vpath`*).
`-rowmapgen` *`configfile deployment outputfile`*	Generates row map cache files. The *`configfile`* setting specifies the name of the DataDeploy configuration file, which can be a standalone configuratin file or one that is generated by running `-intitial` or `-ddgen`. The *`deployment`* setting specifies the name of the deployment you want to use to generate the row map. The *`outputfile`* setting specifies the full path where you want to write the file. Row map cache files can be located in any location that DataDeploy can access during deployments. If the file already exists, it is overwritten. Example: `-rowmapgen medical_dd.cfg basearea /usr/iw-home/datadeploy/conf/medical_rowmap.dat`
`-showbase vpath [dcr-type]`	Shows the base table of the data content record for the specified base path (for example, `/default/main/br/STAGING`). The optional *`dcr-type`* setting specifies displaying a single data type.
`-showdelta vpath [dcr-type]`	Shows the delta table of the data content record for the specified workarea path (for example, `/default/main/br/WORKAREA/wa`). The optional *`dcr-type`* setting specifies displaying a single data type (rather than all data types in *`vpath`*).
`-showtracker`	Shows the tracker table containing all registered tables deployed by DataDeploy.
`-synctracker vpath`	Synchronizes the tracker table for the area named by *`vpath`*. You must execute this option after upgrading from a release of DataDeploy earlier than 4.5.

Option	Description
`-validate vpath [dcr-type]`	Validates `dbschema.cfg` files for data types configured in *`iw-home`*`/local/config/templating.cfg` under the specified workarea *`vpath`*. The optional *`dcr-type`* argument specifies that the `iwsyncdb.ipl` command will validate a single data type's `dbschema.cfg` file (rather than all data types' `dbschema.cfg` files under *`vpath`*). If a particular data type does not have a `dbschema.cfg` file, that type is skipped and no errors are generated.
`-validate dbschema_file_name`	Validates the specified *`dbschema_file_name`*. The *`dbschema_file_name`* argument must specify a complete path to a file that contains the `<dbschema>` element.

Appendix A

Database Server Configuration

Overview

This appendix describes the database server configuration tasks you must perform to configure the following databases to work with DataDeploy:

- IBM DB2 (UDB) 7.1
- Sybase ASE 11.5
- Informix 7.3

IBM DB2

DataDeploy supports IBM DB2 UDB 7.1 on Windows and Solaris systems. The following sections describe how to configure the database server to work with DataDeploy.

Setting Page and Table Sizes

The default pagesize for a tablespace in DB2 is 16K, which is too small for the examples shipped with TeamSite Templating (the examples require that a tablespace of pagesize 32K be already set up on the DB2 server). Also, the default column size and data type used by DataDeploy is `VARCHAR (255)`. These conditions require that you perform one of the following procedures:

1. Make sure that the default tablespace matches the required pagesize (32K). The default tablespace is usually named `IBMDEFAULTGROUP`. Or:

2. Create a tablespace with the required pagesize (32K) and specify the tablespace name as follows in the `<database>` element in the DataDeploy configuration file:

```
<database db = "//host:port/database"
   user = "username"
   password = "password"
   table = "tablename"
   vendor = "ibm"
   tablespace = "tablespacename"
   max-id-length = "anylength">
```

The `tablespace` attribute is valid only for DB2 configuration. It is ignored if you set it when using any other database.

Installing and Starting JDBC

DB2 does not start the daemon to accept JDBC connections by default. You must do this manually by executing the following command:

`db2jstrt` *`port`*

The *`port`* number you enter on the command line must match the *`port`* number shown in the `db` attribute in "Setting Page and Table Sizes" on page 193. If you do not specify a value for *`port`*, it takes a default value of `6789`.

Sybase ASE

DataDeploy supports Sybase ASE 11.5 on Windows NT, Windows 2000, and Solaris systems. The following sections describe how to configure the database server to work with DataDeploy.

Enabling DDL Statements

You must enable DDL statements for transactions as follows. Note that this cannot be done for the master db.

```
1> sp_dboption dbname, "ddl in tran", true
```

Setting Sort Order

Set up case-insensitive sort order for the database by executing the `$SYBASE/bin/sqlloc` utility to set case-insensitive dictionary order. You will also need to recreate indexes on the database that was changed, unless the sort order was changed on initial installation.

Install Stored Procedures

Install jconnect 4.2 stored procedures as follows:

1. Download the jConnect 4.2 package from the Sybase website.
2. Follow the instructions in the "Sybase jConnect for JDBC Installation Guide," Chapter 1, section "Adaptive Server Enterprise" to install the stored procedures for JDBC support into the database.

Informix

DataDeploy supports Informix 7.3 on Windows NT systems, Windows 2000, and Solaris systems. The following sections describe how to configure the database server to work with DataDeploy.

Enabling Logging

Any databases created for use with Informix must be created with logging enabled. This can be accomplished with the Informix tool `dbaccess`, using an SQL command such as the following:

```
create database xyzdb with log
```

Appendix B

Querying Tables

This appendix describes how to query tables through SQL commands that you execute manually after deployment. Methodology differs depending on table type.

Note: You can also embed SQL commands in the DataDeploy configuration file's `<sql>` element. These commands execute automatically during deployment and do not require you to manually query the database. See "Invoking DataDeploy" on page 159 for more information.

Querying Base and Standalone Tables

You can use simple SQL statements specifying key-value pair criteria when querying a base or standalone table. For example:

```
SELECT path FROM staging
    WHERE key = News-Section AND value = Sports;
```

Querying Delta Tables

To query a delta table, you can first create a view consisting of a complex query and then apply a simple query on the view. For example:

```
CREATE VIEW areaview ( key, value, path ) AS
    SELECT key, value, path
    FROM sa
    WHERE NOT EXISTS
           ( SELECT *
                  FROM wa_x WHERE
                       wa_x.key = sa.key AND
                       wa_x.path = sa.path )
    UNION
    SELECT key, value, path
    FROM wa_x WHERE wa_x.state != 'NotPresent';


SELECT path FROM areaview
    WHERE key = News-Section AND value = Sports
```

The `CREATE VIEW` command in this example is the default DataDeploy schema that executes when `table-view` is set to `yes` in the DataDeploy configuration file's `<database>` element.

Appendix C

Event Server

The Event Server component is packaged with TeamSite and provides an efficient, scalable platform for delivering messages (events) to and from your content management applications. Such a platform is a key part in automating content development and deployment tasks.

The Event Server employs advanced technology based on the following principles:

- Reliability — The Event Server stores and queues events, preventing their loss.
- Flexibility — The Event Server enables you to filter events so that applications receive only the ones they need. Additionally, the Event Server is based on Java™ Messaging Service (JMS), a standard API for message delivery.
- Performance — The Event Server's publish-and-subscribe model offers significant performance improvements over the event delivery models used in previous releases that required TeamSite to spawn new processes for each event. Depending on your JMS implementation, asynchronous communication, thread pooling, and database connection pooling also contribute to fast and flexible performance.

Note: If you want to use DAS in a non-U.S. English environment, DAS must use the Event Server.

How the Event Server Works

The Event Server uses the JMS model of message delivery. That model is based on three key concepts:

- *Events* — Synonomous with *message*. Events are the result of changes, end-user actions, or occurences in a Publisher program. For example, TeamSite server events include (but are not limited to):
  - `CreateBranch`
  - `Submit`
  - `SyncDestroy`

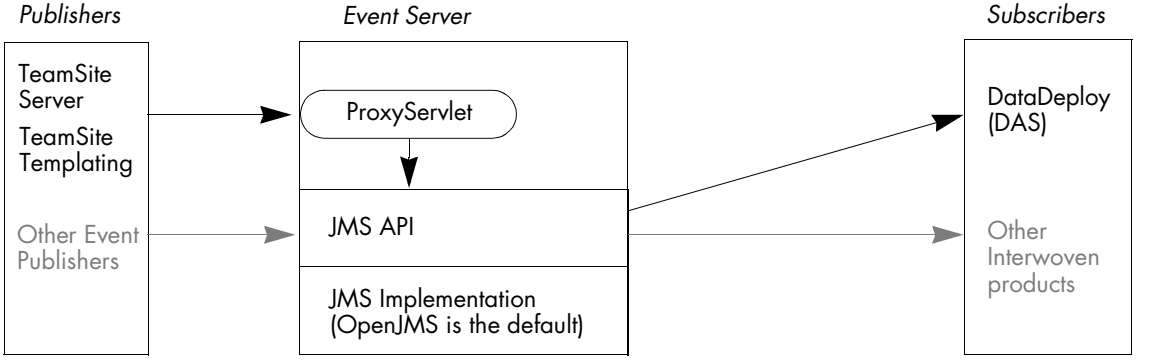
For a list of TeamSite events see "TeamSite Event Triggers" on page 185.

Events have names and properties, such as `user`, `role`, and `timestamp`, that are represented in the Event Server as attribute/value pairs.

Some applications can be configured to perform functions when an event in a different application is triggered. For example, DataDeploy can be configured to run DAS when end users submit content to TeamSite staging areas.

- *Publishers* — Applications that send events to the Event Server. The Event Server then passes the events to Subscribers that have registered to receive them.
- *Subscribers* — Applications that register with the Event Server to receive events. Subscribers can filter events so that they receive only the ones they need.

The following illustrates how the Event Server works (grey items will be supported in future releases):



Supported Applications

This release supports TeamSite as an event Publisher and DataDeploy as a Subscriber. Future releases will support the use of other Interwoven products with the Event Server.

Supported Databases

Interwoven supports the use of DAS with the Event Server when deploying content to the following databases:

- Microsoft SQL Server™
- Sybase® SQL Anywhere®
- Oracle® 8i

Prerequisites

Meet the following prerequisites before using the Event Server:

- TeamSite 5.5 or greater is installed and properly licensed.
- DataDeploy 5.5.1 is installed on the same system where TeamSite is installed.
- DataDeploy is configured to use DAS.
- JDBC 2.0 compatible driver.

Note: Use JDBC type 4 drivers from i-net software if you are using Microsoft SQL Server 2000. The i-net driver is packaged with DataDeploy and is located in *dd-home*`/lib/UNA2000.jar`.

Installing and Enabling the Event Server

The Event Server component is installed with TeamSite.

To enable the Event Server:

1. Open *iw-home*`/etc/iw.cfg` in a text editor of your choice (WordPad is recommended).
2. Find the `[event_subsystem]` section and add the following line:

   ```
   es_enable=true
   ```

   Note: It is important that the value is "`true`" and not "`yes`".

Configuring the Event Sever to Work with DAS

To configure the Event Server to work with the DataDeploy DAS module, do the following:

- Enable the Event Server.
- Set up a database for event persistence.
- Set up Event filters (optional).

Setting Up a Database for Event Persistence

Event persistence must be managed by a RDBMS.

To set up RDBMS-based persistence:

1. Copy:
   iw-home`/eventsubsystem/conf/jmsconfig_rdbms.xml.example`

   to:
   iw-home`/eventsubsystem/conf/jmsconfig.xml`.

2. In the `<DatabaseConfiguration>` section of that file, remove the comment tags from the `RdbmsDabaseConfiguration` section that corresponds to the RDBMS that you want to use.

   Here is an excerpt that shows the commented `RdbmsDabaseConfiguration` section for the Sybase SQLAnywhere database:

```
<!-- Sybase SQLAnywhere example
        <RdbmsDatabaseConfiguration
            driver="com.sybase.jdbc2.jdbc.SybDriver"
            url="jdbc:sybase:Tds:localhost:2638"
            userName="dba"
            password="sql"
            retries="5"
            timeout ="2000" />
-->
```

   Change the values for the `driver`, `url`, `username`, and `password` attributes according to your needs.

   For the complete `jmsconfig_rdbms.xml.example` file, see page 207.

3. Set the `jdbc_classpath` variable in the following file to the location of your database vendor's JDBC driver:

   iw-home`/eventsubsystem/conf/eventsubsystem.properties` file

   Examples:

   (Windows) `jdbc_classpath=c:\\drivers\\sybase\\lib\\SybDriver55.jar`

   (Solaris) `jdbc_classpath=c:/var/iw-home/datadeploy/lib/SybDriver55.jar`

4. Create and register database tables before you start the Event Server. A number of SQL scripts that enable you to create and register the tables are included with DataDeploy and are located in:

   iw-home`/eventsubsystem/conf/ddl/create_`*dbvendor*`.sql`.

   Run the script that corresponds with the database you want to use. You can execute the SQL script using a client utility supplied by the database vendor. For example, if you are using an Oracle database, use SQL*Plus. If you are using Microsoft SQL Server, use Query Analyzer.

5. Start the Event Server.
   - (Windows) Select **Control Panel > Services > InterwovenEventSubsystem**.
   - (Solaris) Run the *iw-home*`/private/bin/iweventsubd` script.

6. Restart `JmsProxyServlet` to ensure that the servlet engine starts publishing to the Event Server.

   ```
   % iwreset
   % iwreset -ui
   ```

Setting Up Event Filters for DAS

You can set up filters so that DAS receives only the TeamSite events that you want it to receive. Specify event filters in *dd-home*/conf/daemon.cfg.

DataDeploy translates the filtering criteria you specify into an SQL statement which it uses to subscribe to the Event Server for TeamSite events.

If no filters are specified, DAS automatically implements a timestamp filter based on the time that DAS started. That is, if no filters have been established and DAS is started, DAS receives all events published since it was started and ignores all previous events.

For details about the *dd-home*/conf/daemon.cfg file, see "DAS Program and Configuration Files" on page 172.

For a list of TeamSite events, see "TeamSite Event Triggers" on page 185.

Sample Filter Section in daemon.cfg

The following is a sample filter:

```
<filter name="EventsFilter">
<keep>
 <and>
       <!-- <field name="timestamp" format="notused" match="now" />
       -->
       <field name="timestamp" format="mm-dd-yyyy hh:mm:ss" match="08-01-
2001
       10:30:00" />
    <in>
       <field name="name" match=" ('Submit', 'CreateWorkarea',
'SyncCreate',
       'DestroyWorkarea')" />
    </in>
 <or>
    <like>
       <field name="user" match="_ob" />
    </like>
 <or>
```

```
    <field name="role" match="master" />
 </or>
    </and>
</keep>

<discard>
 <and>
    <like>
       <field name="area" match="%default%" />
    </like>
 </and>
</discard>
</filter>
```

Note the <keep> and <discard> sections in the sample filter above. Those sections contain the filtering criteria. The <keep> section contains the rules for filtering events that must be processed by DAS. The <discard> section contains the rules for filtering events that do not need to be processed by DAS.

Both sections can contain the following subsections that represent boolean and SQL operators:

- <and> — Events that satisfy all criteria listed in this section are kept (or discarded if used in the <discard> section).
- <or> — Events that satisfy at least one of the criterion listed in this section are kept (or discarded if used in the <discard> section).
- <in> — Used to create a list-based criterion. Events that match at least one of the listed items satisfies the criterion. For example, the sample filter shown above keeps any of the following events would satisfy the <in> criterion: Submit, CreateWorkarea, SyncCreate, or DestroyWorkarea.
- <notin> — Used to create a list-based criterion for events that are to be excluded. Do not use <notin> in a <discard> section; use <in> there instead.

- <like> — Used to create a wildcard-based criterion. You can use the following wildcard characters when specifying the value for the match attribute in <like> and <notlike> subsections:
  - The % character is used to represent any sequence of characters
  - The _ character is used to represent any single character.

  For example, note the <like> subsections in the sample filter shown above:

  Only a three character user name where the last two characters are ob would satisfy the first <like> criterion, and a TeamSite area name of any length that includes the string default would satisfy the second <like> criterion.
- <notlike> — Used to create a wildcard-based criterion for events that are to be excluded. Do not use <notin> in a <discard> section; use <like> there instead.

Each section (or subsection) must contain at least one name and match attribute/value pairs. The name attribute refers to the property name. The value of the match attribute specifies the characteristics of the property that you want to use as a filtering criterion.

DataDeploy would translate that filter into the following SQL statement:

```
( timestamp > 10040500050851 and name IN ('Submit', 'CreateWorkarea', 'SyncCreate',
'DestroyWorkarea') and user like '_ob' or role = 'master' and area NOT LIKE
'%default%')
```

A Note About Filtering Events by Timestamp

By default DAS ignores all events published prior to its start time. If you want DAS to receive events that were published before it was started, establish a timestamp filter. Specify the time from which you want events in the match attribute of the field element. For example, assume that DAS was started on 08/02/2001. Using the sample filter above, DAS would receive events published on and after 10:30:00 the previous day.

Another way to ignore events published before DAS is started is to specify the format and match attributes as notused and now, respectively. See the commented timestamp field in the sample filter above for an example.

The jmsconfig_rdbms.xml.example File

It is recommended that you do not edit the portions in gray.

```
<?xml version="1.0"?>
<JmsConfiguration>
    <ServerConfiguration
        serverClass="org.exolab.jms.server.mipc.IpcJmsServer"
        jmsServerName="OpenJmsServer"
        serverPort="3030"
        serverAddress="localhost" />

    <LoggerConfiguration
        type="ConsoleLogger"
        fileName="openjms.log"
        logLevel="debug" />

    <LeaseManagerConfiguration
        sleepTime="300" />

    <RmiRegistryConfiguration
        embeddedRegistry = "false"
        rmiRegistryPort = "1099" />

    <JndiClientConfiguration
        contextFactory="org.exolab.jms.jndi.mipc.IpcJndiInitialContextFactory">
        <JndiContextFactoryProperty property="org.exolab.jms.jndi.port"
            type="integer" value="3035" />
        <JndiContextFactoryProperty property="org.exolab.jms.jndi.host"
            type="string" value="localhost" />
    </JndiClientConfiguration>

    <JndiServerConfiguration
        jndiServerClass="org.exolab.jms.jndi.mipc.IpcJndiServer"
        jndiServerName="JndiServer"
        jndiServerAddress="localhost"
        jndiServerPort="3035" />

    <ConnectionFactories>
        <ConnectionFactory
            jndiName="JmsQueueConnectionFactory"
            factoryClass="org.exolab.jms.client.JmsQueueConnectionFactory"  />
        <ConnectionFactory
            jndiName="JmsTopicConnectionFactory"
```

```
            factoryClass="org.exolab.jms.client.JmsTopicConnectionFactory"  />
    </ConnectionFactories>

    <MessageManagerConfiguration maxThreads="10"
        destinationCacheSize = "1000"
        garbageCollectionInterval = "300"
        garbageCollectionThreadPriority = "5" />

    <DatabaseConfiguration
        databaseType="rdbms"
        garbageCollectionInterval="180"
        garbageCollectionBlockSize="500"
        garbageCollectionThreadPriority = "5" >
<!-- DB2 (app) setup example
        <RdbmsDatabaseConfiguration
            driver="COM.ibm.db2.jdbc.app.DB2Driver"
            url="jdbc:db2:<dbname>"
     ...
-->
<!-- DB2 (net) setup example
        <RdbmsDatabaseConfiguration
            driver="COM.ibm.db2.jdbc.net.DB2Driver"
            url="jdbc:db2::///<hostname>:<port>/<dbname>"
     ...
-->
<!-- MSSQL setup example
        <RdbmsDatabaseConfiguration
            driver="com.inet.tds.TdsDriver"
            url="jdbc:inetdae7:<hostname>:<port>?database=<dbname>"
     ...
-->
<!-- Oracle setup example
        <RdbmsDatabaseConfiguration
            driver="oracle.jdbc.driver.OracleDriver"
            url="jdbc:oracle:thin:@<hostname>:<port>:<dbname>"
     ...
-->
<!-- Sybase SQLAnywhere example -->
        <RdbmsDatabaseConfiguration
            driver="com.sybase.jdbc2.jdbc.SybDriver"
            url="jdbc:sybase:Tds:localhost:2638"
            userName="dba"
            password="sql"
            retries="5"
            timeout ="2000" />
```

```
    </DatabaseConfiguration>

    <AdminConfiguration
     jmsServer="bin\startjms"
     jmsConfig="conf\jmsconfig.xml"

onlineConnectionMode="org.exolab.jms.administration.mipc.IpcJmsAdminConnection"
        jmsAdminServerName="JmsAdminServer" />

    <AdministeredDestinations>
        <AdministeredTopic topicName="TeamSite_User" />
        <AdministeredTopic topicName="TeamSite_System" />
        <AdministeredTopic topicName="TeamSite_Workflow" />

    </AdministeredDestinations>
</JmsConfiguration>
```

Appendix D

Internationalization

DataDeploy is engineered with your global enterprise in mind. This includes internationalizing DataDeploy to support multibyte languages and locales at the operating system, client, and data management levels. Internationalized DataDeploy supports the following needs:

- Localized operating system—DataDeploy works with any one of the following localized operating systems: English, French, German, and Japanese (one locale per instance of `iwserver`).
- Localized file names—You are no longer restricted to file and directory names in ASCII character encoding. File and directory names can have Japanese names on Japanese servers, German names on German servers, and French names on French servers.
- Continued support for processing of non-English metadata and TeamSite Templating content.

DataDeploy Configuration Files

All configuration files generated by DataDeploy are UTF-8 encoded. Ensure that you save those files as UTF-8 if you edit them. On Windows systems, you can use Notepad or Wordpad to edit those files because those text editors support opening and saving files that are UTF-8 encoded.

DAS in a Non-U.S. English Environment

You must configure DAS to use the Event Server if you want to use DAS in a multibyte environment. See Appendix C, "Event Server" for details about how to use DAS with the Event Server.

OpenDeploy–DataDeploy Synchronization

OpenDeploy–DataDeploy synchronized deployments support multibyte Vpaths.

The -mb option for iwsycdb.ipl

DataDeploy supports an `-mb` option when running `iwsyncdb.ipl`. Use that option if you operate DataDeploy in a non-U.S. English environment; it can also be used for U.S. English locales. The `-mb` option does the following:

- Encodes all configuration files it generates as UTF-8.
- Enables all other `iwsyncdb.ipl` options that use Vpath arguments to be processed by a Java equivalent of `iwsyncdb.ipl`.

Microsoft SQL Server

If you use DataDeploy with Microsoft SQL Server, note the following:

- Use `nvarchar`, `nchar`, and `ntext` instead of `varchar`, `char`, and `text` because Microsoft SQL Server uses Unicode for storing data in `n*` data type columns.
- Override the `CREATE TABLE` statement and use the `nvarchar` data type for columns in the `IWTRACKER` table. Refer to the following file for details:

  dd-home`/conf/create_table_overrides.xml.example`

  You can copy that file to:

  dd-home`/conf/create_table_overrides.xml`

  DataDeploy examines *dd-home*`/conf/create_table_overrides.xml` for an override `CREATE TABLE` statement before it creates `IWTRACKER`, `IWOV_IDMAPS`, and `IWDELTRACKER` tables. If one is not specified, DataDeploy uses `varchar` for text data type columns.
- You must use a case-insensitive and accent-insensitive Unicode collation when you create target databases for deployment.

IBM DB2 Specific Information

The following information is applicable only for systems where the database codeset is different from that of the operating system where the DB2 server is running.

DAS Mode

If you use DAS to deploy the files whose Vpaths contain multibyte characters, you must do one of the following:

- Run the DB2 server on an operating sytem locale that is similar to the muliti-byte code set used to create the target database.
- Set the `DB2CODESET` environment value to the muli-byte code set used to create the target database. Consult page 241, Appedix D: National Language Support, of the *DB2 Administration Guide* for details about how to set the `DB2CODESET` environment value.

Example

Scenario:

- DB2 is running on a US English Windows NT 4.0 platform.
- A database has been created where Japan is specified as the territory and IBM-943 as the code set.
- TeamSite, TeamSite Templating, and DataDeploy are all running on a Japanese language Windows NT 4.0 platform.
- Vpaths of files targeted for deployments contain multibyte store, branch, and workarea names.

Solution:

Configure the DB2 server to use the code set 943 by running the following command:

```
db2set DB2CODEPAGE=943
```

Standalone Mode

You must set the `DB2CODESET` environment value to the muli-byte code set used to create the target database to do any of the following:

- Specify table and column names that contain multibyte characters.
- Specify table and column names that do not contain multibyte characters and deploy multibyte content stored in files targeted for deployment.

The DataDeploy Administration GUI

The DataDeploy administration graphical user interface (GUI) does not support multibyte characters. It is recommended that you do not install the DataDeploy administration GUI on non-U.S. English operating systems.

Test Environments

Interwoven has tested multibyte functionality in DataDeploy with the following database encodings and locales:

Database	Encoding or Locale
Oracle	UTF-8. Deployed German and French data.
	CP1252 and UTF-8.
	JA16EUC (Japanese character set in Oracle).
Microsoft SQL Server 2000	CP1252 and UTF-8.
	Deployed multibyte content to `nvarchar`, `nchar`, and `ntext` data types to a SQL Server 2000 U.S. English database
	Tested German, French, and Japanese collations.
DB2	CP1252 and UTF-8.
	Tested German, French, and Japanese collations.

Miscellaneous

DataDeploy can deploy multibyte content to databases that do not use multibyte character sets but that do support multibyte data types for columns.

When viewing DataDeploy log files that have been created in a multibyte environment, ensure that you use an appropriate text editor to view them.

Index

D